Generación de Narrativa Basada en Criterios de Calidad Aprendidos Automáticamente

Iván Manuel Laclaustra Yebes

MÁSTER EN INGENIERÍA EN INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de fin de Máster en Ingeniería Informática

13 de julio de 2016

Directores: Gonzalo Méndez y Carlos León

Generación de Narrativa Basada en Criterios de Calidad Aprendidos Automáticamente

Iván Manuel Laclaustra Yebes

Trabajo de fin de Máster en Ingeniería Informática, facultad de informática, Universidad Complutense de Madrid

Convocatoria: Junio de 2016
Nota obtenida: 7

Directores: Gonzalo Méndez y Carlos León

Autorización de difusión

Iván Manuel Laclaustra Yebes

13 de julio de 2016

El abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Generación de Narrativa Basada en Criterios de Calidad Aprendidos Automáticamente", realizado durante el curso académico 2015-2016 bajo la dirección de Gonzalo Méndez y Carlos León, en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Agradecimientos

Este proyecto se ha llevado a cabo en el marco de los proyectos WHIM (*grant n.* 611560) y PROSECCO (*grant n.* 600653) financiados por la Comisión Europea dentro del 7º Programa Marco en el área de ICT (*Information and Communication Technologies*) y el programa FET de Futuras Tecnologías Emergentes.

También quisiera agradecer a Pablo Gervás la oportunidad de participar en el proyecto WHIM y a mis tutores, Carlos y Gonzalo, la dirección de este proyecto, así como a Alan Tapscott y el resto del equipo WHIM por su ayuda en el día a día.

Resumen

En la sociedad actual, tenemos en alta estima a aquellas personas que demuestran tener un alto grado de creatividad, ya que implica la correcta aplicación de habilidades cognitivas que comúnmente consideramos reflejo de inteligencia. Con este proyecto, intentamos arrojar algo de luz sobre la creatividad computacional, concentrándonos en el ámbito de la generación automática de historias. Veremos los diferentes paradigmas existentes para la generación automática de historias, así como los sistemas previamente implementados, que nos han servido de referencia para completar el nuestro propio. Expondremos los detalles de nuestro sistema, un primer prototipo de un generador de historias capaz de valorar sus propios resultados, de forma que pueda filtrarlos para presentar los que considera mejores, además de poder cambiar completamente el contexto y el contenido de la historia de forma sencilla. Para ello, nos hemos basado en la generación de historias basada en gramáticas formales, con filtrado basado en un modelo generado previamente mediante aprendizaje máquina. Dicho modelo es generado a partir de las valoraciones de usuarios reales a historias generadas por el sistema para, posteriormente, analizar qué elementos de esas historias son los que desencadenan dicha valoración. De esta forma, estamos estudiando qué elementos hacen que una historia sea interesante para una persona, lo cual es especialmente interesante debido a la falta de consenso en este ámbito. A la hora de narrar las historias, hemos utilizado un enfoque basado en plantillas predefinidas por simplicidad, ya que la generación de lenguaje natural queda fuera del ámbito de este proyecto. Con todo esto, hemos conseguido implementar un generador de historias básico capaz, no sólo de generar un número muy elevado de historias diferentes, si no de valorar cuáles de esas historias son interesantes.

Palabras clave

Creatividad computacional, Gramática formal, Métricas, Aprendizaje máquina, Generador de historias

Abstract

In nowadays society, we hold in high regard those people who demonstrate a high level of creativity, since it implies a correct application of cognitive skills that are commonly considered intelligent. With this project, we're trying to shed some light over computational creativity, focusing on the automatic story generation field. We'll have a look at the different existing paradigms for automatic story generation, as well as the systems previously implemented, that have served as reference for ours. We'll expose the details of our system, a first prototype of a storyteller capable of assessing it's own results, as well as being able to easily change the context and content of the stories generated. For that, we relied on story generation based on formal grammars, with interest level based filtering. This model is generated from the appreciations of the stories by real users to, later on, analyze which elements of the stories trigger these appreciations. This way, we're studying which elements make a story interesting to a real person, being that specially interesting due to the lack of consensus in that field. When telling the stories, we used a template based approach for the sake of simplicity, because natural language generation is out of the scope of this project. With all this, we've implemented a basic story generator capable of, not only generating a high number of different stories, but able to assess which stories are the most interesting.

Keywords

Computational creativity, Formal grammar, Metrics, Machine learning, Story generator

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este capítulo sirve como introducción al contexto de nuestro trabajo. En él, hacemos una pequeña introducción al problema que tratamos, presentando el concepto de creatividad computacional y sus aplicaciones. Posteriormente, podemos ver las razones que nos motivaron a escoger este tema, junto con la metodología seguida. Por último, vemos los objetivos que nos planteamos al comienzo del proyecto, junto con un pequeño resumen de la estructura de la presente memoria.

1.1. Introduction

Until the invention of writing, our only means of inter-generational communication were stories and narrative. This is because stories help us recall the concepts they contain (Sadoski y col., 1990), as we have a natural tendency of thinking, perceiving and imagining in a narrative way (Young y Monroe, 1996). For this reason, stories invite us to use our imagination when listening to them, attracting our attention and strengthening learning. For this, it would be very interesting to make computers communicate with us using stories, specially in fields like education. However, we encounter the problem that a story is a highly intellectually complex product. In them, the author makes use of a wide set of cognitive skills, such as knowledge attribution to certain characters, identification of the goals of each character, validating character's plans, attributing feelings to characters, etc (Gervás, 2009).

Nevertheless, although stories attract our attention naturally (Young y Monroe, 1996), it's clear that not all of them do it in the same degree. Some of them have something that makes us feel unconsciously attracted to them. This "something" varies depending on what we expect to obtain in the story. For example, scary stories have deaths, suspense or tension, while funny stories have awkward situations, misunderstandings, falls, etc. These elements, and the way they're combined, are the ones that make us like (or not) a story. When automatically generating stories, it's important to introduce some valuation method that allows us estimate the quality of each story, so the system is able to improve and present better stories to the end user. That makes us ask ourselves, what makes a story interesting?

In a story, we find several different elements like knowledge, goals, characters with their personality, emotions and feelings, or narration among others (Gervás, 2009). These elements, together with the way they're combined, decide the reader's interest. It's not trivial to find out which of these elements (or possible combinations) contribute to the interest level, although if we knew it'd be "very easy"to generate interesting stories automatically. Therefore, our job consists in recognizing which elements should appear and how to combine them to make interesting stories. As engineers, we have the great advantage of having tools that allow us to have different points of view that we wouldn't have in any other way. For example, the use of intelligent agents, programs situated in an environment capable of acting autonomously on it to satisfy their own goals, is very extended in the field of automatic storytelling, as it helps the characters being completely independent, allowing us to know what they're doing throughout the story.

However there is a problem, as when speaking about creating a story in real life, author's creativity appears to be a main part of the process. The Real Academia Española de la Lengua defines creativity as the çapacity of creating", being that the capacity of .[es]tablishing, funding, introducing something for the first time; giving it birth or live, in a figurative sense"(Real Academia Española, 2014), which implies it being something "newïn some way in the case of artistic creations (such as stories). This, by itself, constitutes a whole research branch: computational creativity. Besides, as we've already seen, the definition of creativity is in itself fuzzy and, in fact, it's hard to explain what it is in our own words. For all this, it's impossible for us to mimic it at a real persons level.

In other way, although most of the existing storytelling systems include some form of evaluation for the stories they generate (León y Gervás, 2010), it's always very attached to the specific domain of the application, so it's hard to extract global knowledge about interest. On the other hand, there have been several attempts on identifying what makes a general story interesting in literature studies, but there's no consensus nowadays. This presents us a difficulty when trying to reproduce creativity in the context of story generation, in the sense that we cannot fulfill the requirements of an interesting story if we do not know them beforehand.

When doing the research work for this project, we realized that the way we tell the story has an measurable impact on the interest we show (McDaniel y col., 2000). This represents a big challenge, as it's difficult to simulate things such as time (necessary, otherwise everything would happen instantly) or conversations (they have to be fluid, spontaneous). In the same way, there are actions that lack of interest by themselves, but can obtain it in combination with another. Eating or sleeping are actions that may not appear in the final story, but they have to if the character at issue meets its enemy while doing them. Besides, it may occur (in fact, it happens) that the characters barely interact, or the result of these interactions lacks interest, but this is an inevitable part of the process.

Inside the process of story generation, we find different approaches (see Sistemas previos de generación de historias, in page 20). Among the most important to take into account we find: plan based, where we have a planner responsible of, given the goals

of the story, decide how it would develope, knowing the features of the characters; rule based, where we have a set of probabilistic rules that will execute depending on the story state, and simulation based, that focuses on generation by recreating the facts that happen. In our case, we'll follow a rule approach, using a formal grammar.

1.2. Introducción

Hasta la invención de la escritura, nuestro único medio de comunicación intergeneracional fueron las historias y la narrativa. Esto es debido a que las historias nos ayudan a recordar mejor los conceptos que contienen (Sadoski y col., 1990), ya que existe una tendencia natural a pensar, percibir e imaginar en forma de narrativa (Young y Monroe, 1996). Por esta razón las historias nos invitan a utilizar nuestra imaginación a la hora de escucharlas, atrayendo nuestra atención y reforzando el aprendizaje. Por ello, sería muy interesante conseguir que los ordenadores se comuniquen con nosotros utilizando historias, especialmente en campos como la educación. Sin embargo, nos encontramos con el problema de que una historia es un producto intelectualmente muy complejo. En ella, el autor hace uso de un amplio abanico de habilidades cognitivas, como la percepción del espacio y el tiempo, la atribución de conocimiento a personajes concretos, identificación de objetivos de cada personaje, validar los planes que los personajes siguen para cumplir sus objetivos, atribuir sentimientos a los personajes, etc (Gervás, 2009).

No obstante, aunque las historias atraen nuestra atención de manera natural (Young y Monroe, 1996), está claro que no todas lo hacen por igual. Algunas tienen algo que hace que nos sintamos atraídos hacia ellas inconscientemente. Ese algo varía en función de lo que esperamos obtener de la historia. Por ejemplo, en las historias de miedo habrá muertes, suspense o tensión, mientras que en una historia de risa habrá situaciones embarazosas, malentendidos, caídas, etc... Estos elementos, además de la forma en que se cuentan, son los que hacen que una historia nos guste o no. Durante la generación automática de historias, es importante introducir algún método de valoración que nos permita estimar la calidad de cada historia para que el sistema pueda mejorar y presentar mejores historias al usuario final. Esto nos hace preguntarnos, ¿qué otorga interés a una historia?

En una historia, nos encontramos con diferentes elementos (Gervás, 2009), como conocimiento, objetivos, personajes con su personalidad, emociones y sentimientos o narración, entre otros. Dichos elementos, junto con la manera en que se combinan, determinan el nivel de interés del lector. No es trivial reconocer cuáles de esos elementos (o posibles combinaciones de los mismos) aportan interés a una historia aunque, si pudiéramos, sería "muy sencillo" generar historias interesantes de manera automática. Por tanto, nuestra tarea consiste en reconocer qué elementos deben aparecer y cómo combinarlos para generar historias de interés. Como ingenieros, tenemos la gran ventaja de poder usar herramientas que nos permitan tener distintos enfoques de la historia que de otra forma no lograríamos. Por ejemplo, en el ámbito de la generación de historias está muy extendido el uso de agentes inteligentes (Wooldridge, 2002), programas situados en un

entorno capaces de actuar autónomamente en el mismo para satisfacer sus propios objetivos, ya que facilita en gran medida que los personajes sean totalmente independientes unos de otros, pudiendo saber qué hace cada uno de ellos en todo momento de la historia.

Sin embargo existe un problema, ya que cuando hablamos de crear una historia en la vida real suele aparecer la creatividad del autor como parte principal del proceso. La RAE (Real Academia Española, 2014) define la creatividad como la "capacidad de crear", siendo esto la capacidad de "establecer, fundar, introducir por vez primera algo; hacerlo nacer o darle vida, en sentido figurado", lo que en el caso de las creaciones artísticas (como las historias) implica que lo creado sea "novedoso" en cierto sentido. Esto, por sí mismo, constituye una rama de investigación dentro de la inteligencia artificial: la creatividad computacional. Además, como hemos podido observar, la misma definición de lo que es la creatividad es difusa y, de hecho, es difícil explicar lo que es con nuestras propias palabras. Es por todo esto por lo que, actualmente, nos es imposible imitarla al nivel de una persona. Por otro lado, aunque la mayoría de los sistemas existentes de generación de historias incluye alguna forma de evaluación para las historias que genera (León y Gervás, 2010), siempre está demasiado sujeta al dominio específico de la aplicación, por lo que es difícil extraer conocimiento global acerca del interés. Por contra, en el campo de estudios literarios ha habido muchos intentos de identificar aquello que da interés a una historia de forma más general, pero no hay ningún consenso actualmente. Esto nos presenta una dificultad a la hora de reproducir creatividad en el contexto de la generación de historias, en el sentido de que no podemos cumplir con los requisitos de una historia interesante si no los conocemos de antemano.

Haciendo el trabajo de investigación necesario para este proyecto, nos dimos cuenta de que la forma en que contamos la historia tiene un impacto apreciable en el interés que mostramos por la misma (McDaniel y col., 2000). Esto representa un gran reto, ya que es complicado simular cosas como el tiempo (es necesario, ya que si no las cosas se producirían instantáneamente) o las conversaciones (deben ser fluidas, espontáneas). De igual forma, hay acciones que carecen de interés por sí mismas, pero que pueden obtenerlo en combinación con otras. Comer o dormir son acciones que pueden no aparecer en la historia final, pero deben hacerlo en caso de que, por ejemplo, el personaje en cuestión se encuentre con su enemigo mientras come. Además, puede ocurrir (y de hecho ocurre) que haya historias sin ningún interés, en las que los personajes apenas interactúen entre ellos, o que el resultado de esas interacciones no tenga ningún fondo, pero esto es una parte inevitable del proceso.

Dentro del proceso de generación de historias, nos encontramos con diferentes enfoques (ver Sistemas previos de generación de historias, en la página 20). Entre ellos los más importantes a tener en cuenta son: basado en planificación, en el que se tiene un planificador que será el encargado de, dados los objetivos de la historia, decidir cómo se desarrollará, conociendo las características de los personajes; basado en reglas, en el que se tienen una serie de reglas probabilísticas que se ejecutarán en función del estado de la historia y de cierta probabilidad; basado en simulación, que se concentra en la generación

mediante la recreación de los hechos que se producen. En nuestro caso, seguiremos el camino de las reglas utilizando una gramática formal.

1.3. Interés y atención

El aprendizaje humano es un proceso selectivo, muy sesgado hacia ciertas características de la información. La mayoría de ellas están presentes en situaciones en las que podemos obtener una experiencia directa. Está aceptado comúnmente que estas experiencias atraen nuestra atención e interés de manera natural, y existen estudios que demuestran que los conceptos aprendidos de esta forma están mejor definidos y son más resistentes al cambio que los aprendidos mediante textos enunciativos (Young y Monroe, 1996). Sin embargo, en muchas ocasiones es imposible obtener experiencia directa, ya sea por cuestiones económicas o de imposibilidad material. Una solución a este problema es el aprendizaje mediante historias, que supone un paso intermedio entre la experiencia directa y la textualización enunciativa. Idealmente, una historia nos hará ponernos en el lugar de su protagonista, imaginando las situaciones por las que debe pasar a lo largo de esta, así como sus sentimientos. No obstante, para que esto funcione es imperativo que la historia que estamos leyendo tenga interés para nosotros, lo cual (junto con otros factores) ayudará a que mantengamos la atención de principio a fin.

Todo lector sabe que un buen libro no requiere de nuestro esfuerzo para su lectura. Cuando leemos una historia que es buena a nuestro modo de ver, nos sentimos atraídos automáticamente hacia ella sin esfuerzo, de forma que toda nuestra atención está ocupada imaginando los detalles que se describen en la misma. La atención es el medio mediante el cual nos concentramos en unos estímulos concretos, ignorando el resto (Young y Monroe, 1996). De ella existen dos tipos: involuntaria y voluntaria. La atracción involuntaria incluye la atracción natural, como sonidos fuertes, los metales o cosas en movimiento, y la adquirida según nuestros gustos, como un tipo de música concreto. Por otro lado, la atención voluntaria (o dirigida) es aquella que requiere una focalización explícita por nuestra parte. Por ejemplo, es el tipo de atención que utilizamos durante nuestras horas de trabajo.

La diferencia más importante entre estos tipos de atención es el esfuerzo que suponen: mientras que la atención involuntaria se produce de forma natural, la atención voluntaria requiere un esfuerzo mental constante por nuestra parte. Por ello, este tipo de atención es difícil de mantener, y nuestra capacidad de concentración disminuye conforme pasa el tiempo. Resulta evidente pues que para que una historia sea satisfactoria es necesario que utilice nuestra atención involuntaria, para lo cual es necesario que nos parezca interesante desde el principio.

1.4. Creatividad computacional

Como sociedad, valoramos mucho a las personas creativas al igual que sus aportaciones a nuestra cultura. De hecho, la creatividad en una persona indica que el individuo posee un amplio abanico de habilidades que consideramos inteligentes, por lo que nos supone un gran desafío técnico reproducir este comportamiento. Dado que la definición de creatividad es demasiado general, no existe un consenso acerca de a qué nos referimos con la creatividad computacional. Decimos que un sistema computacional es creativo en caso de que, asumiendo sus propias responsabilidades, exhiba comportamientos creativos a ojos de observadores imparciales (Colton y Wiggins, 2012). Por tanto, enmarcamos la creatividad computacional en el ámbito de la inteligencia artificial.

La definición anterior hace hincapié en algunos aspectos importantes. El sistema debe asumir responsabilidades, es decir, debe tomar sus propias decisiones durante el proceso creativo, seleccionando por sí mismo los elementos que forman el resultado final. Así, quedan fuera del alcance de la definición aquellos programas que sirven para crear, pero que no toman la iniciativa en ningún momento, como editores de imagen, audio o vídeo. Por otro lado, el sistema debe mostrarse creativo a usuarios imparciales. Esta frase no es trivial, ya que existe una tendencia generalizada (Eigenfeldt, Burnett y Pasquier, 2012) a pensar que las máquinas no pueden ser creativas. Por tanto, en la mayoría de los casos se hace necesaria una evaluación a ciegas del sistema, aplicando una metodología similar al test de Turing.

También es importante notar que la creatividad computacional difiere de la inteligencia artificial general en un aspecto importante: no resuelve problemas, en el sentido de que no es común considerar la creación de una pintura o la composición de una sonata como un problema a resolver. En las creaciones artísticas, la complejidad no es necesariamente sinónimo de valor artístico, por lo que no se puede equiparar un sistema capaz de crear mejores obras artísticas con el tiempo con uno capaz de resolver cada vez problemas más complejos. Es por ello que las técnicas comunes en la resolución problemas de inteligencia artificial como la optimización y la clasificación no aplican directamente. Sin embargo, es evidente que debemos utilizar técnicas de IA conocidas en la medida de lo posible, ya que de esta forma no sólo ahorraremos tiempo, si no que el funcionamiento de ese componente del sistema será óptimo, además de que su uso puede dar lugar a mejoras en la técnica utilizada en el mismo. De hecho, es bastante común combinar diferentes técnicas de inteligencia artificial en el ámbito de la creatividad computacional, haciendo que el sistema final sea capaz de hacer mucho más que la suma de sus componentes (Bundy, 2007). Por ejemplo, en nuestro anterior proyecto (Laclaustra y col., 2014) combinamos el uso de agentes inteligentes con planificación, mientras que el proyecto actual se basa en el uso de gramáticas formales y aprendizaje automático.

Dado que en los sistemas de creatividad computacional no es imperativo que la complejidad computacional aumente con cada problema, se hace necesario un método de evaluación de los resultados que nos ayude a controlar el progreso de nuestro sistema.

Sin embargo, la valoración de creaciones artísticas no es trivial. En el caso de proyectos muy maduros, existen medidas claras en relación a los resultados. Por ejemplo The Painting Fool (Colton, 2012), programa que realiza dibujos abstractos, ha producido resultados que se han vendido en galerías de arte; o The Continuator (Pachet, 2002), programa que improvisa melodías de jazz, que ha improvisado con músicos de jazz en directo. Sin embargo en la mayoría de los casos debemos enfocar la evaluación de una manera más formal, y debemos distinguir entre la evaluación del valor artístico de la obra y la sofisticación del proceso de realización de la misma. Como resultado, la evaluación mediante un "test de Turing" es lo más común, comprobando si el usuario es capaz de distinguir si la obra ha sido creada por una máquina o un humano.

Este último método de evaluación tiene la desventaja de estar muy ligado a la percepción humana del arte. A la hora de crear, las personas estamos limitadas implícitamente por ciertos factores: carácter, personalidad, estado de ánimo, conocimientos sobre el área de trabajo y sobre el mundo que nos rodea, etc. La creatividad computacional ataca directamente este punto, ya que posee una ventaja significativa respecto de las personas: no tiene límites, lo que hace que estos sistemas sean capaces de producir resultados que sorprenden a la audiencia. Por esta razón, al hacer una evaluación a ciegas estamos excluyendo aquellos resultados en los que podemos ver que el autor es una máquina, pero que pueden tener cierto valor artístico. Como ejemplo, The Painting Fool utiliza un algoritmo evolutivo para ordenar rectángulos en un lienzo, produciendo imágenes similares al skyline de Manhattan. Para ello se utilizó en primera instancia una función de ajuste hecha a mano, proporcionada de manera externa al programa. Como experimento, se proporcionaron los conceptos con los que se había implementado dicha función a un programa encargado de encontrar nuevas funciones de ajuste, que se proporcionaron a The Painting Fool más adelante. En uno de los resultados obtenidos los edificios estaban colocados unos sobre otros, lo cual hace caso omiso de las leyes físicas, pero es muy interesante artísticamente.

1.5. Motivación

A la hora de la generación de historias, resulta interesante la generación basada en gramáticas formales por varias razones. En primer lugar, el tiempo de ejecución es mucho menor que utilizando otros enfoques (como el basado en agentes inteligentes), además de conseguir un conjunto de todas las posibilidades que podemos generar en cada momento. Por otro lado, es muy sencillo aumentar el número de historias a generar, simplemente añadiendo nuevas reglas, lo que lo hace muy expandible. Por último, su uso era muy favorable en nuestro caso debido a que los archivos de entrada a utilizar nos obligaban a seguir una estructura muy estricta para las historias.

En cuanto al filtrado según el interés, resulta muy conveniente el uso de técnicas de aprendizaje automático. Esto se debe a que no están claros los elementos que otorgan interés a una historia y, utilizando esta técnica, no nos es necesario conocerlos de antemano. La utilización de usuarios reales hace que los elementos que el modelo considera

como interesantes sean consistentes con datos reales, mejorando su porcentaje de acierto. Por otro lado, el cálculo de los atributos que se utilizarán para entrenar nuestro modelo de aprendizaje automático, debe ser también automático, ya que el número de historias es demasiado elevado como para realizarlo manualmente. Esto hace que nuestro sistema sea adaptable a distintos generadores de historias, de forma que podemos añadir nuevos atributos para encontrar los que más peso tienen sobre la medida de interés.

1.6. Metodología

A continuación, listamos los pasos que seguimos para la consecución del proyecto. Aunque se realizó un cambio de enfoque (ver Cambio de paradigma, en la página 35) a lo largo del mismo, el procedimiento general no varía.

1. Implementación del generador de historias: ya sea mediante agentes inteligentes (como era en un principio) o con gramáticas formales, el primer paso a seguir es implementar un generador que nos proporcione las historias sobre las que trabajar.
2. Generación de los conjuntos de datos: utilizando las historias proporcionadas por el generador, debemos crear dos conjuntos de datos: el conjunto de entrenamiento, mediante el cual generamos el modelo y el conjunto de test, que utilizamos para calcular el porcentaje de acierto que obtenemos.
3. Parametrización de los conjuntos de datos: una vez obtenidos los conjuntos de datos, debemos implementar funciones que transformen automáticamente las historias que contienen en datos objetivos (atributos), de forma que puedan ser utilizados por el modelo.
4. Valoración con usuarios: debemos utilizar usuarios reales para valorar qué historias tienen interés y cuáles no lo tienen, de forma que el etiquetado de las instancias de los conjuntos de entrenamiento y test se base en opiniones reales. Así, nuestro modelo realizará predicciones más precisas.
5. Generación del filtro: a partir del conjunto de entrenamiento, entrenamos un modelo de predicción del interés de las historias, que actúa como filtro de las mismas. Dado que en un principio no sabemos la distribución que tendrá el interés de las historias, debemos entrenar nuestro modelo basándonos en diferentes algoritmos. Mediante el conjunto de test, podemos calcular el porcentaje de acierto de cada uno. Nuestro modelo será aquel con el porcentaje de acierto más alto.

1.7. Objetivos

Nuestro objetivo es generar un modelo de interés en una historia, aprendido automáticamente. Para ello, utilizaremos un enfoque basado en gramáticas para la generación del conjunto completo de historias y, posteriormente, filtraremos las historias en función

del nivel de interés predicho por el modelo, descartando las que no sean interesantes. Una vez terminado, transformamos el conjunto de hechos de cada historia filtrada en texto en lenguaje natural, mediante la aplicación de plantillas predefinidas.

Durante el transcurso del proyecto nuestros objetivos se vieron ligeramente alterados. Concretamente, se cambió el enfoque de generación de las historias, pasando de generar historias extensas a historias cortas, pero en mayor número, obteniendo el conjunto completo de posibilidades. A causa de esto, el modelo a generar también varió ligeramente, ya que en lugar de filtrar las partes más interesantes de una historia era necesario clasificar la historia completa.

A continuación, expondremos uno por uno los objetivos propuestos al comienzo del proyecto:

- Diseñar un generador de historias capaz de crear historias extensas y diferentes.
- Generar un modelo de interés aprendido automáticamente, que dé como resultado las partes más interesantes de la historia proporcionada.
- Conseguir que el sistema sea capaz de mejorar a partir de la interacción con el usuario, mejorando el modelo de filtrado con cada interacción.

Tras el cambio de enfoque nuestros objetivos pasaron a ser los siguientes:

- Diseñar un generador de historias capaz de crear un amplio conjunto de historias cortas y diferentes.
- Generar un modelo de interés aprendido automáticamente, que dé como resultado una clasificación de la historia proporcionada en función de si es o no interesante.
- Conseguir que el sistema sea capaz de mejorar a partir de la interacción con el usuario, mejorando el modelo de filtrado con cada interacción.

1.8. Estructura del documento

A continuación, podemos ver una breve descripción del contenido de cada capítulo:

- En el capítulo 1 hacemos una introducción, describiendo el problema a tratar junto con nuestra propuesta.
- En el capítulo 2 echamos un vistazo al trabajo previo existente en el ámbito de nuestra solución, viendo los conceptos y herramientas necesarios para su comprensión. También introducimos nuestro marco de trabajo, así como los sistemas previos de generación en los que nos basamos.
- En el capítulo 3 podemos ver el diseño del sistema a nivel de componentes, para hacernos una idea del funcionamiento general.

- En el capítulo 4 pasamos a ver los detalles de nuestra implementación, viendo a nivel de código cómo hemos estructurado el programa, así como las decisiones que se han tomado en cada momento.
- En el capítulo 5 evaluamos los resultados que hemos obtenido y discutimos acerca de los puntos fuertes y débiles de nuestro proyecto.
- En el capítulo 6 ofrecemos las conclusiones acerca del resultado final. También podemos ver algunas líneas de investigación por las que consideramos sería interesante continuar.

Capítulo 2

Trabajo relacionado

En este capítulo hacemos una breve introducción al trabajo previo relacionado con nuestro área de trabajo. Comenzamos poniendo en contexto el proyecto actual, enmarcado en el ámbito del proyecto europeo WHIM. Posteriormente, introducimos los conceptos de gramática y aprendizaje automático, donde podemos ver las diferentes opciones que tenemos para su utilización. Por último, hacemos un pequeño resumen de algunos de los generadores de historias existentes, de los que hemos tomado inspiración.

2.1. Marco de trabajo: WHIM (The What-If Machine)

La *What-If Machine* (WHIM, FET grant number 611560[1]) es un proyecto de ámbito europeo que comenzó en octubre del año 2013, en el que participan equipos de cinco universidades. Fué creado como un proyecto STREP (short-term research project, proyecto de investigación a corto plazo) por el programa FP7 de la Comisión Europea como parte del objetivo ICT-2013.8.1: Tecnologías y fundamentos científicos en el campo de la creatividad.

Se encarga de generar *What-Ifs*, frases del tipo "¿Qué pasaría si...?" en inglés. Por ejemplo, *"What if birds aren't singing but are actually screaming because they are afraid of heights?"*. Uno de los componentes del proyecto es un generador de historias que, a partir de un What-If dado, genera una historia alrededor de los puntos clave de la misma.

El proyecto WHIM es un intento de desarrollar un primer estudio a gran escala acerca de cómo un software puede inventar, evaluar y expresar ideas ficticias de valor. Existen muchas formas de automatizar la generación de ideas, e inventarlas no es una tarea sencilla, ya que requiere mucho conocimiento del mundo y del dominio. Por ejemplo, en el *What-If* de más arriba, debemos saber que los pájaros están normalmente a gran altura, que los pájaros cantan, que cantar es distinto de gritar, que los pájaros son animales, que los humanos somos animales, que la gente en ocasiones tiene miedo de las alturas, etc.

[1] http://www.whim-project.eu/

2.1.1. Mini Narrative

Una de las características más interesantes del proyecto WHIM es la capacidad de cambiar por completo el contexto de los *What-if* que genera. Cada *What-if* genera un archivo de salida llamado *Mini Narrative*, necesario para el cálculo de determinadas métricas (entre ellas, el número total de historias que se pueden generar con él).

Una *mini narrative* es el resultado de un *What-If*, y contiene una lista de hechos primarios (*narrative points*) que deben suceder en la historia. Mediante la variación de estos hechos, podemos cambiar completamente el contexto de la historia, así como los hechos que deben aparecer en la misma.

2.1.2. Participantes

Este proyecto tiene un coste total de 2.193.614 €, y en él participan las siguientes entidades:

- Goldsmiths' College - Reino Unido (**Coordinador**)
- University College Dublin, National University of Ireland, Dublin - Irlanda
- The chancellor, Masters and Scholars of the University of Cambridge - Reino Unido
- Universidad Complutense de Madrid - España

2.2. Gramática

Desde el día que nacemos estamos expuestos a nuestro idioma materno por medio de nuestra inclusión en la sociedad. Gracias a ello, por un método de prueba y error, somos capaces de aprender nuestro idioma de manera intuitiva, de forma que, después, somos capaces de hablar de corrido sin necesidad de pensar en cómo se construye una frase o palabra. No obstante, existen ciertas reglas que están presentes de manera implícita, y nuestro uso del lenguaje las respeta de manera natural. La formalización de estas reglas constituye lo que hoy en día conocemos comúnmente como gramática.

Aunque la escritura ha estado presente con nosotros desde su invención en Mesopotamia en el 3.200 a.C, los primeros intentos de encontrar un sistema común de formación del lenguaje llegaron con la Grecia clásica, impulsados por la expansión del imperio de Alejandro Magno, debido a la gran cantidad de lenguas que en él se hablaban. Sin embargo, fué a partir de la creación de la primera gramática española en 1.492 por Antonio de Nebrija cuando el interés en la gramática aumenta. Esta corriente es la que a partir del siglo XIX da lugar a la lingüística moderna y, a partir del siglo XX, a las gramáticas formales.

Podemos ver un ejemplo muy claro del uso de las gramáticas a la hora de aprender un segundo idioma. Inicialmente, es necesario que aprendamos de manera más formal las reglas de construcción de frases que utiliza, así como el vocabulario necesario. Conforme vamos aprendiendo, esas reglas son interiorizadas, de manera que no tenemos que pensar en ellas para utilizarlas.

2.2.1. Gramáticas formales

En teoría de lenguajes, llamamos gramática formal a un conjunto de reglas necesarias para la producción de cadenas de caracteres pertenecientes a un lenguaje concreto (Chomsky, 1972). Estas reglas describen cómo formar cadenas utilizando el conjunto de símbolos disponibles del lenguaje (alfabeto) que encajen con la sintaxis de dicho lenguaje, sin entrar en su significado, lo que hace que las gramáticas sean muy utilizadas tanto como generadores como reconocedores de lenguaje.

Según la descripción formal dada por Chomsky en 1956 (Chomsky, 1956) una gramática posee los siguientes componentes:

- Un conjunto finito N de símbolos no terminales, asociados con la parte izquierda de las reglas.
- Un conjunto finito Σ de símbolos terminales, es decir, el alfabeto.
- Un conjunto finito P de reglas de producción, que tendrán al menos un no terminal en su parte izquierda.
- Un símbolo $S \in N$ que se utilizará como regla inicial.

A partir de la regla inicial, comenzamos a aplicar reglas en función de sus apariciones dentro de la regla actual. La ejecución de reglas termina en el momento en que la cadena generada contiene únicamente símbolos terminales. El conjunto de todas las cadenas que podemos generar es lo que llamamos lenguaje. Dado que podemos tener varias posibles producciones para una misma regla, debemos seguir todos los caminos posibles para generar el lenguaje completo.

El uso de gramáticas (Gervás, 2013) para la generación de historias tuvo su auge durante los años 70 y principios de los 80, como consecuencia del trabajo realizado por Chomsky en el campo de las gramáticas formales. Durante los años posteriores surgió un debate sobre su uso en el campo de la generación de historias, el cual dió como resultado la desacreditación de las gramáticas como un modelo de proceso cognitivo utilizado realmente a la hora de la generación (Black y Wilensky, 1979), además de su abandono en el ámbito de la generación de historias, en favor de técnicas como la planificación. No obstante, a raíz del trabajo de Propp (Propp, 1968) a comienzos del siglo 20 (en el que llega a la conclusión de que las historias folclóricas rusas son muy estructuradas) surgieron propuestas de gramáticas inspiradas en su trabajo, ampliándolo con otras técnicas (por ejemplo, Grasbon y Braun (Grasbon y Braun, 2001) añadieron funciones polimórficas con diferentes salidas en función de la interacción del usuario). En nuestro

caso, seguiremos un enfoque parecido, aplicando una nueva técnica tras la generación, ya que el uso de gramáticas nos proporciona una serie de ventajas que resultan interesantes para el proyecto WHIM (ver Marco de trabajo: WHIM (The What-If Machine), en la página 11).

Cuando Chomsky formalizó por primera vez las gramáticas en 1956 (Chomsky, 1956), estableció una jerarquía que las divide en varios subtipos. Cada uno de ellos es más restrictivo que el anterior, lo que hace que el conjunto de lenguajes que se pueden expresar con ella sea menor. No obstante, tienen una serie de ventajas que pueden ser importantes en función de lo que queramos conseguir (por ejemplo, estos tipos más restrictivos son muy utilizados en compiladores, ya que su nivel de complejidad es menor, lo que permite un análisis eficiente del texto). Estos tipos son:

- Sin restricciones (Tipo 0).
- Dependientes de contexto (Tipo 1): gramáticas en las que la parte izquierda de las reglas posee un símbolo no terminal, rodeado por cadenas de símbolos terminales, mientras que en la parte derecha sólo tenemos símbolos terminales.
- Libre de contexto (Tipo 2): gramáticas en las que la parte izquierda de las reglas contiene un sólo símbolo no terminal.
- Regular (Tipo 3): además de las restricciones del tipo 2, la parte derecha de estas reglas debe contener: la cadena vacía, un terminal o bien un terminal seguido de un no terminal.

En nuestro caso, al no necesitar cadenas complejas en la parte izquierda de las reglas, utilizaremos gramáticas libres de contexto. Puesto que no necesitamos analizar frases dentro del lenguaje, no necesitamos optimizar el análisis, y no utilizaremos una gramática regular que, además, limitaría nuestras posibilidades a la hora de generar historias.

2.2.2. Lenguajes de programación para gramáticas

Para el generador de historias necesitamos un lenguaje capaz de expresar gramáticas formales de un modo sencillo, además de darnos rápidamente todas las posibilidades que la gramática puede formar. Para ello resulta conveniente la utilización de un lenguaje de programación lógica o declarativa. En este caso los principales candidatos son Prolog, Haskell y Maude. Prolog es un lenguaje ubicado dentro del paradigma de la programación lógica, muy utilizado en el ámbito de la IA. Maude es un lenguaje declarativo pensado para realizar auditorías, por lo que funciona de manera óptima a la hora de explorar el conjunto de soluciones de un problema, mientras que Haskell es un lenguaje funcional multipropósito. Una desventaja a tener en cuenta de este tipo de lenguajes es que resultan poco intuitivos, por lo que es necesario un periodo de adaptación mayor que con los lenguajes imperativos tradicionales. Sin embargo, ofrecen mayor flexibilidad, ya que en ellos no especificamos cómo queremos que se encuentre una solución. Dado que Prolog es un lenguaje antiguo (la primera versión es de 1971) y ampliamente utilizado en el ámbito

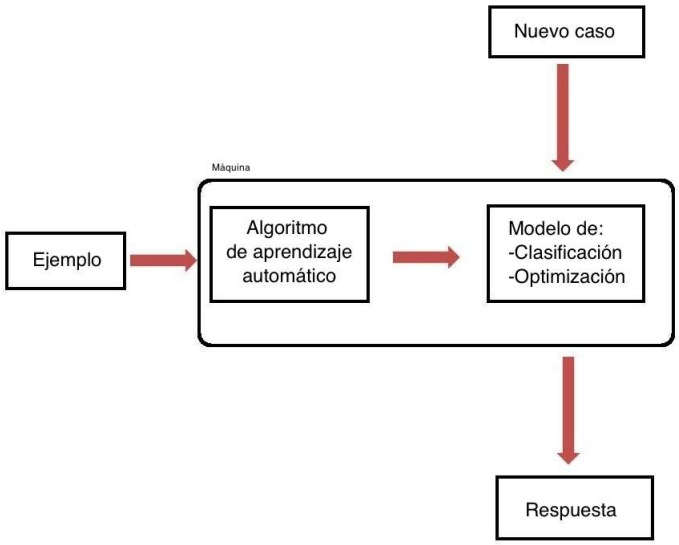
de la IA, existe gran cantidad de documentación y ejemplos. Además, es el lenguaje más versátil de los presentados, permitiendo técnicas como el encaje de patrones más complejos que los otros lenguajes. Por todo ello, llegamos a la conclusión de que Prolog es la mejor opción en nuestro caso.

2.3. Aprendizaje automático

Como hemos dicho, a la hora de la creación de una historia entran en juego en gran medida conceptos como creatividad o interés, cuya definición es demasiado general. Esto nos supone un gran problema a la hora de generar historias con interés para el lector, ya que, utilizando algoritmos secuenciales convencionales, sería necesario incluir los elementos interesantes explícitamente en el modelo. Además, es muy común en la vida real que estos elementos no estén del todo claros para el lector, que símplemente sabrá si le interesa la historia o no. Por todo esto, se hace necesario un clasificador que sea capaz de distinguir qué elementos son los más importantes a partir de la clasificación de la historia (dada por una persona real).

Actualmente considerada una rama de la inteligencia artificial, Arthur Samuel definió (Simon, 2013) en 1959 el aprendizaje automático como el "campo de estudio que proporciona a un computador la capacidad de aprender sin ser explícitamente programado". Este campo explora la construcción de algoritmos que puedan aprender y hacer predicciones a partir de un conjunto dado de datos (Samuel, 1959). Dichos algoritmos construyen un modelo del tipo de datos que se proporciona, que permite tomar las decisiones oportunas ante nuevas instancias de datos, dependiendo del tipo de algoritmo escogido.

Figura 2.1: Esquema del proceso de aprendizaje automático



El aprendizaje automático se utiliza en un abanico de tareas en las que no es favorable utilizar algoritmos secuenciales convencionales. Algunas de las más conocidas incluyen filtrado de spam, reconocimiento de caracteres, motores de búsqueda o conducción autónoma. En muchos casos, esta disciplina se solapa con la minería de datos, consistente en la búsqueda de información y patrones dentro de conjuntos de datos.

Esta disciplina ofrece un amplio conjunto de algoritmos, por lo que debemos escoger cuidadosamente el que mejor se adapte a nuestras necesidades. Cada algoritmo pertenece a una de las siguientes clases:

- Supervisado: el conjunto de datos incluye, para cada instancia, su etiqueta de clase. De esta forma, el sistema puede aprender las características de cada clase, pudiendo predecir después instancias nuevas.
- No supervisado: tenemos un conjunto de datos sin etiquetado, a partir del cual debemos distinguir las distintas clases existentes, sin entrar en qué significa cada una. Esto se realiza a partir del análisis de las características de cada instancia.

Cada una de estas clases presenta una serie de ventajas e inconvenientes, haciendo que el aprendizaje automático se adapte bien a diferentes tipos de problemas. Por ejemplo, el aprendizaje no supervisado se utiliza principalmente para la detección de anomalías, es decir, encontrar aquellos sujetos que no pertenecen a un grupo general (un uso típico de esta técnica es encontrar productos defectuosos en una cadena de producción). En cuanto al aprendizaje supervisado, suele usarse cuando queremos clasificar instancias en función de clases concretas, como por ejemplo el reconocimiento de caracteres escritos a mano, donde debemos proporcionar al sistema no sólo la forma del caracter, si no el propio caracter al que hace referencia. En nuestro caso, utilizamos un enfoque supervisado para la clasificación de nuestras historias, ya que nuestro objetivo es etiquetarlas como interesantes o no en función de valoraciones reales.

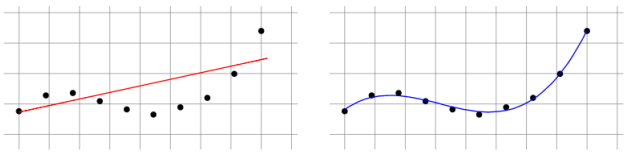
Dentro del aprendizaje supervisado nos encontramos con múltiples algoritmos. Por brevedad, presentaremos los algoritmos que aplican a nuestro trabajo, aunque algunos de ellos tienen también aplicación dentro del aprendizaje no supervisado. Aunque existen gran cantidad de ejemplos ampliamente utilizados en la práctica (k vecinos más cercanos, redes neuronales, deep learning o clustering entre otros) expondremos sólo aquellos que son relevantes para nuestro proyecto.

2.3.1. Regresión

El conjunto de algoritmos de regresión hace uso de una función matemática que intenta aproximarse a la distribución de la nube de puntos que forman los datos de aprendizaje. Funciona para instancias con cualquier cantidad de atributos (gráficamente, cada atributo es una dimensión) y para conjuntos de datos numéricos. De esta forma, podemos calcular fácilmente una aproximación de la clase (numérica) de las instancias nuevas, que se corresponde con la salida de la función calculada previamente, una vez introducidos los datos de la nueva instancia.

Será necesario probar con diferentes tipos de función para diferentes tipos de conjuntos de datos, ya que tendrán diferentes distribuciones. Para ello, es necesario calcular diferentes modelos, y calcular el error medio que se obtiene en su aplicación.

Figura 2.2: Regresión lineal vs regresión polinómica: las instancias poseen dos dimensiones (atributo y clase). Introduciendo una de las variables, podemos aproximar la otra. (Shalev-Shwartz y Ben-David, 2014)



2.3.2. Support Vector Machines

Las *Support Vector Machines* fueron introducidas en el año 1995 (Cortes y Vapnik, 1995), y son un conjunto de métodos relacionados para el aprendizaje supervisado que sirven tanto para regresión como para clasificación, y están especialmente pensados para conjuntos de datos con alta dimensionalidad de atributos (Maletic y Marcus, 2010). Este paradigma se concentra en separar el conjunto de datos mediante una línea (o hiperplano) que esté lo más alejada posible de la instancia más cercana de cada clase. Esta circunstancia hace que su complejidad de cómputo sea baja incluso para espacios dimensionales altos. Este paradigma admite diferentes funciones mediante las que separar el conjunto de datos tanto para el caso de regresión como para clasificación, haciéndolo más versátil. Por brevedad nos centraremos en la utilidad de clasificación.

Este tipo de algoritmos se basan en la premisa de que el riesgo de que el hiperplano escogido sea incorrecto es inversamente proporcional al margen, la distancia mínima entre una instancia y la superficie de decisión.

El caso más sencillo con el que nos podemos encontrar es la clasificación de instancias con dos clases, separables por una línea recta. En este caso, encontrar el hiperplano de separación se plantea como un problema de optimización: encontrar una función que separe correctamente ambos conjuntos de datos maximizando el margen. Esta función tendrá como parámetros los atributos de las instancias utilizadas durante el entrenamiento. Una vez obtenida, le proporcionamos los atributos de las nuevas instancias y observamos el signo del resultado para diferenciar las clases.

Sin embargo, en la mayoría de los casos limitarnos a un clasificador lineal es simplificar demasiado las cosas. Por ello, se hace indispensable poder cambiar la función que caracteriza al hiperplano generado. Esto se realiza mediante una técnica conocida como el "truco del *kernel*" (Cortes y Vapnik, 1995), que consiste en aplicar una función no lineal al conjunto de atributos proporcionados, dando como resultado un nuevo

Figura 2.3: Funcionamiento de las *Support Vector Machines*. El resultado nos dará el plano central. (Shalev-Shwartz y Ben-David, 2014)

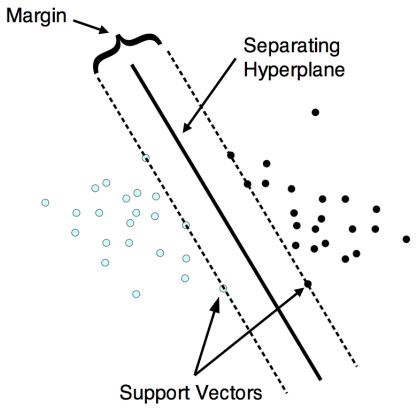
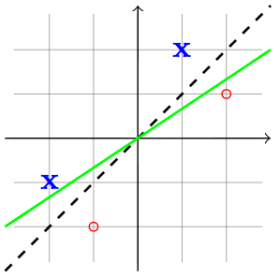


Figura 2.4: Ambos hiperplanos separan las instancias correctamente. El SVM nos dará el de línea discontinua. (Shalev-Shwartz y Ben-David, 2014)



conjunto de mayor dimensionalidad. Una vez hecho esto, el problema se reduce a calcular el hiperplano que separa las clases de las instancias utilizando el nuevo conjunto de dimensionalidad superior.

No obstante, la elección de cuál es la función más apropiada a la hora de transformar el conjunto de atributos es un campo de estudio (Chen, Lin y Schölkopf, 2005) en sí mismo. Podemos ver algunas de las funciones que más se utilizan en el tabla 2.1.

Para los casos en los que nos conjuntos no son totalmente separables mediante un hiperplano, se introducen parámetros adicionales a la función del *kernel*. Estos parámetros sirven para rebajar las restricciones de los márgenes. De esta forma, pueden existir instancias que no estén correctamente clasificadas, dando como resultado un modelo menos preciso, pero más aplicable en la vida real.

Figura 2.5: Utilizando una función no lineal, transformamos el conjunto de atributos en uno de mayor dimensionalidad para separar las clases. (Shalev-Shwartz y Ben-David, 2014)

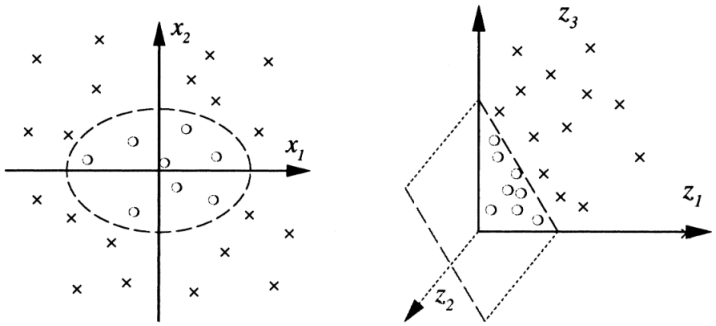


Tabla 2.1: Algunas de las funciones más utilizadas como *kernels* en SVM. (Shalev-Shwartz y Ben-David, 2014)

Kernel	$K(x, x_i)$
Función de base radial	$exp(-\gamma\|\|x - x_i\|\|^2), \gamma > 0$
Multicuadrática inversa	$\frac{1}{\sqrt{\|\|x-x_i\|\|+\eta}}$
Polinómico de grado d	$((x^T \cdot x_i) + \eta)^d$
Sigmoidal	$\tanh(\gamma(x^T \cdot x_i) + \eta), \gamma > 0$
Lineal	$x^T \cdot x_i$

2.3.3. Herramientas para aprendizaje automático

En cuanto al módulo de aprendizaje automático existen diferentes opciones. Dado que existen implementaciones genéricas de los algoritmos de aprendizaje automático más utilizados, sencillamente debemos encontrar un sistema que los incluya y ejecute de manera correcta. Para asegurarnos de esto, buscamos aquellos programas que incluyeran la librería LibSVM (Chang y Lin, 2011), diseñada para proporcionar herramientas de clasificación del tipo SVM, así como regresión y estimación de distribuciones. De esta búsqueda, surgen 2 principales candidatos: Weka y R. Ambos programas sirven para el propósito de nuestra aplicación. R es un entorno para la computación estadística y la creación de gráficos ampliamente utilizada, mientras que Weka es una herramienta diseñada para la minería de datos en Java, más específica y ligera.

Weka (Hall y col., 2009) es un proyecto desarrollado por la Universidad de Waikato (Nueva Zelanda), y consiste en una colección de algoritmos de aprendizaje automático para tareas de minería de datos. Estos algoritmos pueden ser aplicados directamente sobre un conjunto de datos dado a partir de la interfaz gráfica que proporciona, o utilizados desde nuestro propio código Java gracias a la librería incluida. Contiene herramientas

para el pre-procesado de datos, clasificación, regresión, clustering, reglas de asociación y visualización, aunque también se adapta de manera satisfactoria al desarrollo de nuevos esquemas y algoritmos de aprendizaje automático. Weka es una herramienta gratuita que opera bajo una licencia GNU General Public License.

R (Ng, 2006) es una herramienta (además de un lenguaje) para la computación gráfica y estadística. Provee una gran cantidad de técnicas, tanto gráficas como estadísticas. Entre éstas últimas se incluyen clasificación, clustering o modelado lineal/no lineal. Una de sus principales ventajas es la facilidad con la que podemos generar gráficas y modificaciones de los conjuntos de datos, además de incluir símbolos matemáticos y fórmulas. La herramienta utiliza su propio lenguaje, aunque es posible añadir fragmentos de código en lenguajes como C, C++ o Fortran. Otra ventaja es que es un sistema extensible mediante paquetes. Tanto los paquetes como la herramienta son gratuitos, sujetos a una licencia GNU General Public License.

Llegamos a la conclusión de que lo mejor para el caso que nos ocupa era utilizar *Support Vector Machines*, debido a varias razones. En primer lugar, es un paradigma que funciona bien para conjuntos de datos de naturaleza muy diferente, ya que nos proporciona diferentes opciones para utilizar en el *kernel*. Además, es capaz de paliar el efecto "mínimo local", por lo que nos aseguramos de que el hiperplano obtenido como resultado es óptimo. Por último, su entrenamiento y test se realiza de forma muy rápida, al contrario que otros modelos.

Nos decantamos por utilizar Weka porque incluye la librería LibSVM, de forma que la parte del aprendizaje automático se reduce a obtener el conjunto de datos (usuarios reales), encontrar los atributos (*features*) más fuertemente relacionados con el interés de una historia y la presentación de nuevas instancias al modelo para su clasificación. Por otro lado, su sencillez de uso (incluye una interfaz gráfica) y la facilidad de incluirlo en código Java nos brindan gran flexibilidad a la hora del cálculo de filtros (ver Capítulo 4 - Modelo de interés de una historia) y de presentar nuevas instancias. Se utilizó la versión Weka 3.6.

2.4. Sistemas previos de generación de historias

Durante la fase de investigación, pudimos ver distintos ejemplos de generadores de historias. Nuestro objetivo durante esta fase era el de hacernos una idea de cómo podíamos aproximarnos al problema. En el sentido de la generación basada en gramáticas pudimos encontrar ejemplos muy significativos. Sin embargo, no hallamos nada relacionado con la predicción del interés utilizando aprendizaje automático.

2.4.1. Novel Writer

Novel Writer (Klein y col., 1973) es el primer generador automático de historias documentado. Utiliza reglas probabilísticas para producir historias de asesinatos, en el contexto de una fiesta de fin de semana. Para generarlas, el sistema se vale del conjunto de reglas para realizar cambios en el estado del mundo introduciendo acciones de los personajes y eventos, que dan lugar al contenido propiamente dicho.

En este primer prototipo, la variedad de las historias que se generan es escasa, ya que la secuencia de escenas está descrita explícitamente en el código, dejando a la simulación únicamente la interacción entre los personajes (de hecho, podríamos considerarla como una primera versión de generación basada en gramáticas). Además, el conjunto de reglas utilizado para la simulación es muy limitado, lo cual da lugar a pocas posibilidades de generación.

Como inputs el sistema recibe una descripción del mundo en el que se ambienta la historia, y una descripción de los personajes que en ella aparecen. Víctima y asesino son elegidos en tiempo de ejecución, en función de las características de los personajes (con un ingrediente aleatorio adicional), mientras que los motivos surgen en función de los eventos surgidos durante el transcurso de la historia.

A continuación, mostramos un ejemplo de historia generada por este sistema (éste es sólo un pequeño episodio de una historia de 2.100 palabras):

The day was Monday. The pleasant weather was sunny. Lady Buxley was in the park. James ran into Lady Buxley. James talked with Lady Buxley. Lady Buxley flirted with James. James invited Lady Buxley. James liked Lady Buxley. Lady Buxley liked James. Lady Buxley was with James in a hotel. James caressed Lady Buxley with passion. James was Lady Buxley's lover. Marion following saw the affair. Marion saw the affair. Marion was jealous.

2.4.2. Talespin

Talespin (Meehan, 1977) es un sistema que cuenta historias sobre las vidas de las criaturas de un bosque. Este sistema introduce por primera vez el concepto de motivación personal de cada personaje. Así, la personalidad de cada uno se modela como una medida ponderada de características numéricas, como la amabilidad, la arrogancia o la honestidad. Los personajes están modelados de forma que puedan tener relaciones complejas, como familiaridad, confianza o afecto, que actúan como precondiciones y postcondiciones en función de la acción concreta.

Utiliza un enfoque basado en reglas para la generación de historias, combinando forward chaining (desde los eventos hasta sus consecuencias) con backward chaining (de consecuencias de un evento previo, que serán los objetivos, a eventos que puedan conducir a esas salidas), en el que se descomponen los objetivos en otros más pequeños

para facilitar la generación. Estos objetivos son añadidos a la lista global de objetivos tras la ejecución de una acción.

En Talespin, la evaluación de las historias es muy simple, ya que la función de evaluación se encarga de comprobar si el objetivo del protagonista se ha cumplido o no. Meehan debate sobre qué hace válida a una historia (existencia de un problema, grado de dificultad de solucionar el problema y naturaleza o nivel del problema resuelto), aunque una evaluación general se sale del ámbito del programa.

A continuación, podemos ver un ejemplo de un extracto de una historia generada:

John Bear is somewhat hungry. John Bear wants to get some berries. John Bear wants to get near the blueberries. John Bear walks from a cave entrance to the bush by going through a pass through a valley through a meadow. John Bear takes the blueberries. John Bear eats the blueberries. The blueberries are gone. John Bear is not very hungry

2.4.3. Author

Author (Dehn, 1981) intenta modelar la mente del autor en el ámbito de la creación de historias. De esta forma el sistema se encarga de que se añadan a la historia episodios memorables, personajes y demás, partiendo de la base de que los hechos sobre el mundo de la historia ya están estructurados. También intenta modelar la mente humana en el sentido de cómo se organiza el conocimiento y cómo se accede a él, siguiendo las teorías de cómo funciona la memoria (Kolodner, 1980) (Schank, 1982).

Dehn sostiene que los eventos que ocurrirán en la historia son la semilla principal. En otras palabras, los eventos son lo primero que se modela, para después construir un mundo y personajes a su alrededor que los justifiquen. A nivel de código, esto se traduce en objetivos relacionados con dichos eventos, como por ejemplo los personajes que participarán en una determinada acción o su rol en la historia. Para la construcción de estos eventos, el autor debe satisfacer otra serie de objetivos encargados de mantener (o intentarlo) la calidad de la historia, como asegurar la plausibilidad, credibilidad de los personajes, consistencia o el grado en el que la historia mantiene la atención del lector. Por tanto, podemos concluir que la reformulación de dichos objetivos durante la generación supone la mayor parte de la carga computacional del sistema. Dehn denomina esto "reformulación conceptual": la idea inicial se reformula en el episodio principal (y eso a su vez en una sucesión de episodios), una caracterización se presenta como un episodio en el que se trata, un cambio en una relación entre personajes se presenta como un diálogo que da lugar a ella, etc.

Para Dehn, la generación de historias es un proceso de razonamiento creativo en el que intervienen dos componentes: deliberado, en el que se utilizan los eventos y objetivos que se tenían en principio, y aleatorio, que hace referencia a las ideas que surgen durante el proceso de generación. Para modelar esto, se añaden dos objetivos a lista de objetivos del autor: cumplir el objetivo narrativo actual y encontrar mejores objetivos a partir

de este. Cuando el sistema encuentra posibilidades prometedoras no contempladas al principio, el rumbo de la historia cambia, añadiendo ese componente de inspiración.

2.4.4. Universe

En Universe (Lebowitz, 1983) los objetivos se modelan en función del tipo de historia y personajes se quieren generar, enmarcando todo ello en el contexto de una sucesión de capítulos de una telenovela, por lo que presenta gran cantidad de personajes interpretando historias múltiples, simultáneas y superpuestas, sin un final concreto. Es el primer generador de historias que presta especial atención a los personajes, utilizando estructuras de datos complejas para representarlos. Propone un simple algoritmo para rellenarlas, aunque la mayor parte debe rellenarse de forma manual, por lo que casi toda la carga de trabajo recae en el usuario.

El objetivo de Universe es la generación de una historia continua, sin principio ni final claramente definidos, dando lugar a historias muy extensas. Sin embargo, el sistema alterna entre planificar la continuación del argumento y contar el argumento generado hasta el momento desde la última vez que se contó algo.

Este sistema funciona mediante fragmentos de argumento, que proporcionan métodos narrativos que cumplen objetivos del autor. Estos fragmentos se utilizan a la hora de la generación, para dar lugar a conflictos entre los personajes, realizando acciones que no forman parte de sus objetivos necesariamente. Para la generación propiamente dicha, se utiliza la planificación descomposicional: el sistema posee un grafo de precedencia con los objetivos de autor pendientes y los fragmentos de argumento, manteniendo una relación entre ellos y los eventos ya contados. La siguiente fase del argumento se produce al expandir uno de los objetivos que cumpla las precondiciones. Las diferentes ramas de la historia (objetivos posibles a partir de un mismo punto) se van expandiendo progresivamente.

El sistema incluye un mecanismo para expandir automáticamente su librería de fragmentos de argumento, creando nuevos fragmentos. Esto se consigue generalizando fragmentos existentes e instanciando la estructura resultante para generar más. De esta forma, el sistema es mucho más extensible, lo cual es importante ya que se implementó con vistas a la ayuda para los escritores, con posibilidades de desarrollarlo como un generador autónomo en el futuro.

Lebowitz argumenta que el mundo de la historia es lo primero que se crea, situando en él los personajes, localizaciones y objetos que se necesiten. Este punto de vista choca diametralmente con el de Dehn, que opina que el argumento es el que dirige la construcción del mundo.

A continuación, ponemos a modo de ejemplo una historia generada por este sistema:

Liz was married to Tony. Neither loved the other, and, indeed, Liz was in love with Neil. However, unknown to either Tony or Neil, Stephano, Tony's father, who wanted

Liz to produce a grandson for him, threatened Liz that if she left Tony, he would kill Neil. Liz told Neil that she did not love him, that she was still in love with Tony, and that he should forget about her. Eventually, Neil was convinced and he married Marie. Later, when Liz was finally free from Tony (because Stephano had died), Neil was not free to marry her and their trouble went on.

2.4.5. Minstrel

Minstrel (Turner, 1993) es un programa que genera historias sobre el rey Arturo y sus caballeros de la tabla redonda. Inicialmente, se proporciona un tema o moraleja que se utiliza como base para las historias, que son de entre media y una página de longitud. Según Turner, Minstrel puede generar unas diez historias de esta longitud, y puede crear una serie de escenas de historias más cortas.

Para la generación, Minstrel utiliza bloques de construcción, que contienen una serie de objetivos (que pueden ser de autor o de los personajes) y planes que los satisfacen. La construcción de historias en Minstrel funciona como un proceso de dos fases, englobando una fase de planificación y una de solución de problemas. La fase de planificación opera sobre los objetivos de autor, guardandolos en la memoria de trabajo. El proceso consume los objetivos de la memoria bien dividiendolos en subobjetivos de autor, que a su vez son guardados de nuevo en la agenda, o bien pasándose a la fase de solución de problemas ,que intenta solucionarlos añadiendo los ingredientes necesarios a la historia. Para el cumplimiento de los objetivos, Minstrel utiliza una técnica llamada Case-Based Reasoning (CBR), que utiliza ejemplos anteriores para razonar. Así, el sistema posee una base de datos con historias previamente generadas sobre diferentes temas. Una vez se presenta un objetivo, el sistema realiza una búsqueda en su base de datos, identificando aquellas historias en las que dicho objetivo estaba presente, para poder generar nuevas escenas para la historia actual. Una vez completada, la historia se añade a la base de datos. Cada vez que se crea una escena, Minstrel la revisa para comprobar si proporciona una oportunidad de aplicar uno de los objetivos a nivel de autor que aseguren consistencia, o introduce uno de los motivos literarios deseados. Esto encaja con el segundo metaobjetivo de Dehn de buscar nuevos objetivos de autor [11].

A continuación, ponemos a modo de ejemplo ilustrativo una historia generada por el sistema:

The Vengeful Princess

Once upon a time there was a Lady of the Court named Jennifer. Jennifer loved a knight named Grunfeld. Grunfeld loved Jennifer. Jennifer wanted revenge on a lady of the court named Darlene because she had the berries which she picked in the woods and Jennifer wanted to have the berries. Jennifer wanted to scare Darlene. Jennifer wanted a dragon to move towards Darlene so that Darlene believed it would eat her. Jennifer wanted to appear to be a dragon so that a dragon would move towards Darlene. Jennifer drank a magic potion. Jennifer transformed into a dragon. A dragon moved

towards Darlene. A dragon was near Darlene. Grunfeld wanted to impress the king. Grunfled wanted to move towards the woods so that he could fight a dragon. Grunfeld moved towards the woods. Grunfeld was near the woods. Grunfeld fought a dragon. The dragon died. The dragon was Jennifer. Jennifer wanted to live. Jennifer tried to drink a magic potion but failed. Grunfeld was filled with grief. Jennifer was buried in the woods. Grunfeld became a hermit.

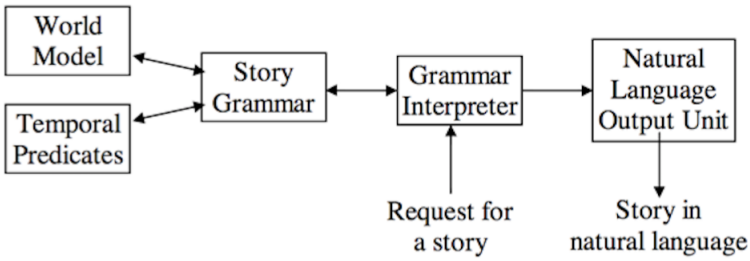
MORAL: Deception is a weapon difficult to aim.

2.4.6. Joseph

Joseph (Lang, 1997) es un sistema generador de historias basado en gramáticas formales, continuando así con el trabajo de Propp en el campo de las historias folclóricas rusas. En este caso, se considera que las historias están formadas por un planteamiento, seguido de una serie de episodios en los que la acción toma lugar. Estos episodios consisten en un hecho, una reacción por parte del protagonista y un desenlace, pudiendo encadenar tantos como se desee.

En este sistema, los elementos terminales de la historia están separados del resto en el modelo del mundo, para mayor modularidad. Una particularidad del generador de Lang es la capacidad de considerar el paso del tiempo a lo largo de la historia, lo que sirve para relacionar los hechos que se suceden. Por otro lado, el generador de lenguaje es el encargado de traducir la lista de hechos que han sucedido a un texto legible.

Figura 2.6: Componentes de Joseph. (Lang, 1997)



Algunos ejemplos de historias que puede generar el sistema son:

once upon a time there lived a dog. one day it happened that farmer evicted cat. when this happened, dog felt pity for the cat. in response, dog sneaked food to the cat. farmer punished dog.

once upon a time there lived a cossack. one day it happened that imp possessed daughter of a boyar. when this happened, cossack felt love for the daughter of a boyar. in response, cossack made it his goal that he would be married to the daughter of a boyar. cossack exorcised the imp from the daughter of a boyar. cossack was married to daughter of a boyar.

2.4.7. Mexica

Mexica (Pérez ý Pérez, Mike Sharples, 2001) es un modelo computacional diseñado para estudiar el proceso creativo de la escritura, en términos de compromiso (engagement) y pensamiento (Sharples, 1999). Está diseñado para generar historias cortas sobre los primeros habitantes de Méjico.

Mexica basa su funcionamiento en 3 estructuras fundamentales: acción, definida en función de sus precondiciones y postcondiciones; historia, formada por un conjunto concreto de acciones y contexto histórico, denominado Story-World Context (SWC), que representan situaciones en las que una acción ha aparecido en una historia previa, y actúan como reglas. Estas situaciones están modeladas en función de las conexiones emocionales y tensiones existentes entre los personajes.

A la hora de añadir acciones a la historia, el sistema funciona en 2 fases: engagement y reflection. Durante la fase de engagement se construye la historia, utilizando las SWC para encontrar acciones concretas que añadir a la historia. De esta forma, las acciones añadidas no son al azar, lo que incrementa el interés general de las historias generadas por el sistema.

La evaluación de las historias se lleva a cabo durante la fase de reflection, y genera una valoración en términos de coherencia, interés e innovación. Para ello, el sistema compara la historia generada hasta el momento con el conjunto de historias previas. Si la historia se parece mucho a otra anterior, o su valoración de interés es baja en relación a la media, el sistema asigna una plantilla que sirve como filtro para las acciones que se pueden añadir a partir de ese momento, de forma que se restringen aún más. Para evaluar el interés, realiza una representación en función de la tensión de la historia, y la compara con el resto de historias del sistema. Una vez encontrada la más parecida, se utiliza como referencia de valoración, por lo que el "gusto" del sistema se basa en las historias que posee. La coherencia se comprueba al finalizar la generación. Se encarga de llevar a cabo ajustes para que la historia tenga sentido, como añadir al texto ciertos objetivos o tensiones explícitamente.

En otras palabras, el sistema añade una acción (que debe estar presente en un SWC que encaje con el argumento) y evalúa la historia hasta ese momento. Al finalizar, el sistema comprueba la coherencia general, y realiza los ajustes finales a la historia.

A continuación, podemos ver un ejemplo de una historia que genera:

Jaguar knight was an inhabitant of the Great Tenochtitlan. Princess was an inhabitant of the Great Tenochtitlan. Jaguar knight was walking when Ehecatl (god of the wind) blew and an old tree collapsed injuring badly Jaguar knight. Princess went in search of some medical plants and cured Jaguar knight. As a result Jaguar knight was very grateful to Princess. Jaguar knight rewarded Princess with some cacauatl (cacao beans) and quetzalli (quetzal) feathers

2.4.8. Virtual Storyteller

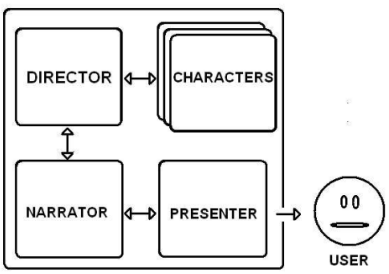
Virtual Storyteller (Theune y col., 2003) sigue un enfoque en el que los personajes construyen el argumento de la historia. En este caso, presenta una aproximación multi-agente para la generación, en la que cada personaje es un agente autónomo inteligente, por lo que actúa en función de su propio conocimiento y de la información que recibe de su entorno, en este caso, el mundo en el que situamos la historia y el resto de agentes que participan. Mediante la recolección de las acciones que llevan a cabo los personajes, el sistema forma una historia. Este enfoque resulta muy interesante para la generación de narrativa interactiva, de forma que las acciones de los agentes dependan de las del usuario, por lo que la aplicación en videojuegos resulta prometedora.

Además de los personajes, el sistema incluye dos agentes de soporte: el agente director, encargado de cuidar el argumento y el agente narrador, encargado de generar el lenguaje natural. El agente director posee información básica acerca de la estructura que debe tener una historia, en este caso, principio, nudo y desenlace, y se encarga de que la historia se ajuste a esta estructura. Además, vela por que la historia sea consistente, en el sentido de que las acciones concuerden con la personalidad de cada personaje. Para asegurar estas dos condiciones, el director puede realizar ciertos cambios sobre la historia: añadir nuevos personajes y objetos, asignar nuevos objetivos a los personajes y desautorizar acciones concretas. Sin embargo no puede forzar a un personaje a realizar una acción concreta, por lo que la generación del argumento recae completamente en los personajes.

El agente narrador es el encargado de traducir la representación de estados y eventos a frases en lenguaje natural. El esfuerzo se concentra sobre todo en la correcta generación de pronombres para hacer que el texto resultante parezca más natural.

Este sistema no posee ningún método de evaluación de las historias generadas por lo que, aunque todas serán consistentes y tendrán una estructura bien definida, no se asegura que la interacción entre los personajes de lugar a una historia con interés para el lector.

Figura 2.7: Arquitectura del Virtual Storyteller



A continuación, podemos ver un pequeño fragmento de una historia generada por este sistema:

Because Princess Lovely heard that the lamb had fled, she was sad. She wanted the lamb back, so she wanted to find it and ran out of the castle. Meanwhile, the hungry children slaughtered the lamb. Because Lovely saw the dead lamb, she was sad and hurt.

2.4.9. Fabulist

Fabulist (Riedl y Young, 2010) es una arquitectura para la generación y presentación automática de historias. Divide el proceso de generación de narrativa en tres niveles: generación de fábula, generación de discurso y representación.

La generación de la fábula usa un enfoque de planificación para la generación de la narrativa. Riedl asegura que la narrativa posee dos elementos que son universales: la progresión lógica causal del argumento y la credibilidad de los personajes, que es la percepción del receptor de que las acciones realizadas por los personajes no impactan negativamente en su suspensión de la incredulidad, la voluntad del sujeto para dejar de lado su sentido crítico, ignorando incoherencias o incompatibilidades de la obra (Schaper, 1978). Por ello, el algoritmo de planificación utilizado (IPOC) razona sobre la causalidad, la intencionalidad y motivación de un personaje simultáneamente, a fin de producir secuencias narrativas causalmente coherentes (en el sentido de que avanzan hacia una conclusión) y posee elementos sobre la credibilidad de los personajes, además de generar argumentos robustos causalmente. Fabulist primero genera un plan narrativo que concuerda con el objetivo de salida, asegurando que todas las acciones y objetivos de los personajes son justificados con eventos dentro de la propia narrativa.

A continuación proporcionamos un ejemplo de historia generada por Fabulist. Los archivos de entrada incluyen: un modelo del dominio que describe el estado inicial del mundo de la historia y las posibles operaciones que los personajes pueden realizar, y un estado final.

There is a woman named Jasmine. There is a king named Jafar. This is a story about how King Jafar becomes married to Jasmine. There is a magic genie. This is also a story about how the genie dies. There is a magic lamp. There is a dragon. The dragon has the magic lamp. The genie is confined within the magic lamp. King Jafar is not married. Jasmine is very beautiful. King Jafar sees Jasmine and instantly falls in love with her. King Jafar wants to marry Jasmine. There is a brave knight named Aladdin. Aladdin is loyal to the death to King Jafar. King Jafar orders Aladdin to get the magic lamp for him. Aladdin wants King Jafar to have the magic lamp. Aladdin travels from the castle to the mountains. Aladdin slays the dragon. The dragon is dead. Aladdin takes the magic lamp from the dead body of the dragon. Aladdin travels from the mountains to the castle. Aladdin hands the magic lamp to King Jafar. The genie is in the magic lamp. King Jafar rubs the magic lamp and summons the genie out of it. The genie is not confined within the magic lamp. King Jafar controls the genie with the magic lamp. King Jafar uses the magic lamp to command the genie to make Jasmine love him. The genie wants Jasmine to be in love with King Jafar. The genie casts a spell on Jasmine making her fall in love with King Jafar. Jasmine is madly in love with King Jafar. Jasmine

wants to marry King Jafar. The genie has a frightening appearance. The genie appears threatening to Aladdin. Aladdin wants the genie to die. Aladdin slays the genie. King Jafar and Jasmine wed in an extravagant ceremony. The genie is dead. King Jafar and Jasmine are married. The end.

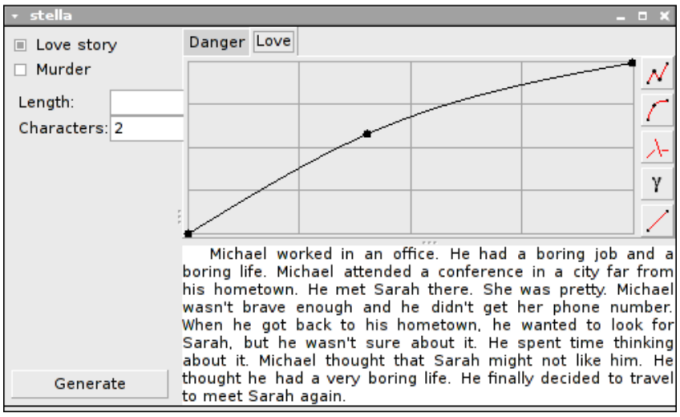
2.4.10. Stella

Stella (León, 2011) es un generador de historias diseñado para ser una herramienta para la generación automática de narrativa. Debido a ello, el motor de generación y el modelado de dominio están aislados, comunicados tan sólo por la interfaz, de forma que el cambio (o ampliación) del conocimiento de dominio es muy sencillo. Para que el sistema pueda funcionar, es necesario proporcionar este conocimiento de dominio de forma externa, lo que hace que dependa en gran medida de dicho conocimiento. Al ser una herramienta pensada para un uso efectivo, no sólo para investigación, si no también para producción, el conocimiento de dominio se ha implementado de forma muy modular, siendo así fácilmente ampliable.

Este sistema basa el proceso de generación en la sucesiva aplicación de las reglas de dominio a la historia inicial, lo que da lugar un conjunto completo de historias que se pueden generar con el dominio. Sin embargo, el sistema explora el árbol durante el proceso de generación, lo que hace que este proceso sea mucho más liviano computacionalmente. Una vez hecho esto, el sistema presenta una de las historias al usuario en función de las restricciones que se han proporcionado en la interfaz gráfica.

En esta herramienta se aplican técnicas para el modelado de personajes, emociones, estructura de la historia y de uso intensivo de conocimiento. La generación del texto se consigue mediante plantillas y correcciones a mano.

Figura 2.8: Interfaz de STellA (León, 2011)



Posteriormente, la herramienta fue ampliada para utilizar simulación además de aplicación del conocimiento de dominio (León, 2014). De esta forma, el conocimiento sobre

el mundo en el que están presentes las historias es mucho más rico en detalles, además de simplificar mucho su extensibilidad. Así, el modelado de los sentimientos, objetivos, foco de atención y demás características de los personajes se hace más sencillo. Para ello, el sistema aplica las reglas del conocimiento de dominio a dos niveles: personajes y mundo. La simulación se produce a la hora de aplicar las reglas al mundo, de forma no determinista.

Un ejemplo de historia producida por el sistema es:

Michael worked in an office. He had a boring job and a boring life. Michael attended a conference in a city far from his hometown. He met Sarah there. She was pretty. Michael wasn't brave enough and he didn't get her phone number. When he got back to his hometown, he wanted to look for Sarah, but he wasn't sure about it. He spent time thinking about it. Michael thought that Sarah might not like him. He thought he had a very boring life. he finally decided to travel to meet Sarah again.

2.4.11. Generación con agentes software y planificación

El sistema (Laclaustra y col., 2014) está compuesto (ver figura 2.9, en la página 31) por un sistema multiagente en el que sucede la historia, un planificador, que es utilizado por los agentes para saber qué hacer en cada momento, y un logger, encargado de recoger los eventos acontecidos durante la historia, de forma que puedan servir de input para un generador de lenguaje natural. En el sistema multiagente, el agente director (a cargo de crear al resto de agentes) y el agente mundo (a cargo de cargar el mapa de localizaciones desde un archivo externo, y mantenerlo actualizado con las posiciones de los personajes) son los encargados de mantener la integridad de la historia. El resto de agentes del sistema (puede haber tantos como se decida) corresponden a los personajes de una historia de princesas y dragones, en la que un dragón secuestra a una princesa y un caballero debe rescatarla. Estos personajes cargan tanto las acciones de las que disponen como los objetivos desde un archivo externo (domain.pddl). Para poder utilizar el planificador, es necesario que el agente en cuestión escriba el estado actual de la historia (a partir del cual se planifica) en un archivo PDDL, que le sirve como entrada (problem.pddl).

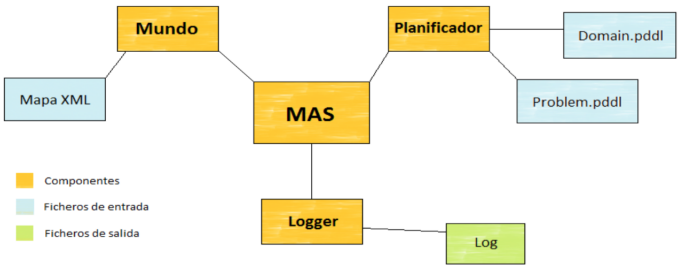
Aquí podemos ver algunos ejemplos de historias que este sistema es capaz de generar:

El Rey Rey está preparado. La Princesa Princesita despierta. La Princesa sale del castillo. El dragón Draco emprende el vuelo en busca de alguna princesa desprotegida. Las siguientes princesas son secuestrables: Princesita. La Princesa Princesita ha sido secuestrada. Intentando pedir rescate para la princesa Princesita. Encontrados los siguientes caballeros: ninguno. El caballero Paquito entra en escena. Enconctrados los siguientes caballeros: Paquito. El dragon Draco ha muerto en batalla. La Prinecsa Princesita fué liberada. El Rey entrega 50 monedas al caballero Paquito. La Princesa llega al castillo con el caballero Paquito. La princesa Princesita pone fin a su aventura.

La Princesa Princesita despierta. El caballero Juanito entra en escena. El Rey Rey está preparado. El dragón Draco emprende el vuelo en busca de alguna princesa desprotegida. El caballero Paquito entra en escena. Las siguientes princesas son secuestrables: Princesita. Draco ha llegado al Castillo. Draco ha secuestrado a Princesita. Princesita ha llegado a la Montaña. Draco ha llegado a la Montaña. La Princesa Princesita ha sido secuestrada. Intentando pedir rescate para la princesa Princesita. Encontrados los siguientes caballeros: Paquito, Juanito. Paquito ha llegado a Montaña. El dragón Draco ha muerto en batalla. El caballero Paquito ha liberado a la princesa Princesita. Paquito ha llegado al Castillo. Princesita ha llegado al Castillo. El caballero Paquito ha dejado a la princesa en su castillo. Paquito ha llegado al Pueblo. El caballero Paquito se ha convertido en héroe. La princesa Princesita fué liberada. El Rey entrega 50 monedas al caballero. La princesa Princesita pone fin a su aventura.

El caballero Juanito entra en escena. El caballero Paquito entra en escena. El Rey Rey está preparado. La princesa Princesita despierta. El dragón Draco emprende el vuelo en busca de alguna princesa desprotegida. Las siguientes princesas son secuestrables: Princesita. Draco ha llegado al Castillo. Draco ha secuestrado a Princesita. Princesita ha llegado a la Montaña. Draco ha llegado a la Montaña. La Princesa Princesita ha sido secuestrada. Intentando pedir rescate para la princesa Princesita. Encontrados los siguientes caballeros: Paquito, Juanito. Paquito ha llegado a la Montaña. El caballero Paquito ha muerto en combate. El Rey recibe la noticia de la muerte del caballero, así que busca otro. Intentando pedir rescate para la princesa Princesita. Encontrados los siguientes caballeros: Juanito. Juanito ha llegado a la Montaña. El dragón Draco ha muerto en batalla. El caballero Juanito ha liberado a la princecsa Princesita. Princesita ha llegado al Castillo. Juanito ha llegado al Castillo. El caballero Juanito ha dejado a la princesa Princesita en su castillo. Juanito ha llegado al Pueblo. El caballero Juanito se ha convertido en héroe. La princesa Princesita fué liberada. El rey entrega 50 monedas al caballero Juanito. La Princesa Princesita pone fin a su aventura.

Figura 2.9: Arquitectura del generador de historias basado en agentes.



2.5. Conclusión

Viendo los trabajos que hemos presentado anteriormente, podemos sacar algunas conclusiones importantes. En primer lugar, notamos que la generación de historias basada en gramáticas es un enfoque que actualmente está en desuso, debido, en parte, a enfoques más sofisticados, como el uso de agentes software.

Por otro lado, resulta interesante notar que, aunque la mayoría de sistemas incluye algún método de valoración de los resultados, siempre está muy ligado al dominio específico de la aplicación. Nuestro sistema ataca directamente este punto, presentando una metodología aplicable a cualquier tipo de generador de historias. Así, nuestro propósito es valorar el interés de nuestras historias de forma más general para, idealmente, extraer conocimiento sobre los elementos de interés de una historia cualquiera.

Capítulo 3

Diseño

En este capítulo veremos la estructura general del sistema, en términos de los componentes que posee y la forma en que interactúan. Explicamos brevemente el cambio de rumbo acontecido durante el proyecto, así como las decisiones de diseño que se tomaron.

3.1. Elementos de una historia

Para que un ordenador sea capaz de contarnos historias, debemos buscar qué elementos forman una historia contada por una persona. El primer elemento son los personajes. Las acciones, relaciones e interacciones entre los mismos dan lugar a una serie de sucesos, que nos servirán de base para la historia a generar. También necesitamos un escenario en el que puedan moverse, lo que dará lugar, a su vez, a nuevas interacciones, esta vez con el entorno. Cuanto más complejo sea éste, mayor posibilidad de interacción con el mismo hay, pero menor posibilidad de interacción entre los personajes, por lo que es necesario encontrar un equilibrio entre el tamaño del mapa y el número de personajes. Estos personajes deben conocer su entorno para poder actuar en consecuencia. Una vez comenzada la simulación de la historia, necesitamos una manera de recoger las acciones de los personajes, de forma que podamos manipularlas durante el proceso de narración. Para dicho proceso, debemos ser capaces de transformar una serie de hechos descritos en un formato concreto, a lenguaje natural reconocible. Por otro lado, para que la historia no sea aburrida, debemos escoger cuidadosamente los eventos que estarán presentes en el resultado final, para lo que se hace necesario saber qué elementos son los que hacen interesante una historia.

Cada una de las historias posee una serie de parámetros que la definen, que pueden ir desde lo más simple, como la longitud o la cantidad de personajes, hasta cosas más complejas, como la tensión presente en las relaciones entre los personajes. Por tanto, los posibles parámetros que podemos escoger no están claros y las posibilidades son infinitas.

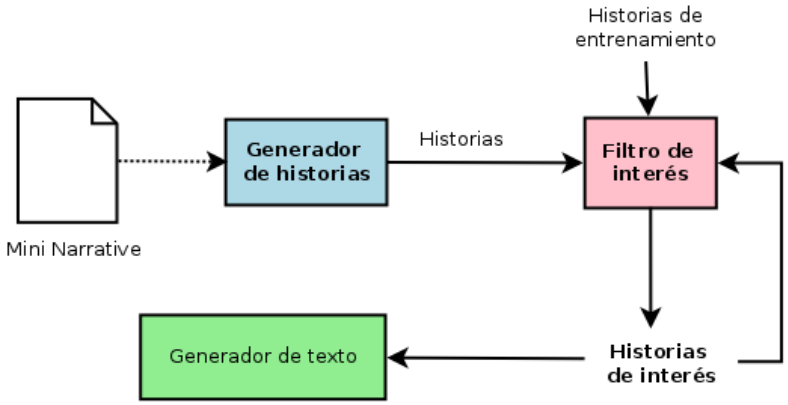
3.2. Arquitectura del sistema

Este proyecto consta de dos partes principales:

- Generador de historias: Es el encargado de crear todas las historias que nuestra gramática es capaz de generar.
- Modelo: responsable de aplicar el filtro a los eventos de la historia.

Opcionalmente, sería interesante añadir un generador de lenguaje natural que transforme nuestras historias (consistentes en un conjunto bruto de hechos) en una historia propiamente dicha. En nuestro caso, hemos optado por la generación de texto basada en plantillas (ver Generación de lenguaje natural, en la página 47) por motivos de sencillez, aunque el uso de un generador de lenguaje natural proporciona texto de mayor calidad.

Figura 3.1: Arquitectura del proyecto



El generador produce el conjunto de historias que se le pasa al modelo de aprendizaje automático generado, que actúa como filtro, proporcionando como salida aquellas historias que lo satisfagan. Dicho modelo debe ser entrenado previamente con un conjunto de historias debidamente etiquetadas.

El modelo de aprendizaje máquina calcula los atributos de cada historia proporcionada por el generador de manera automática, para posteriormente utilizar dichos atributos para la predicción. De esta forma, no sólo podemos clasificar conjuntos muy grandes de historias en muy poco tiempo, sino que es muy fácil añadir nuevos atributos y modificar las historias que generamos. Debemos tener en cuenta que el cálculo de dichos atributos debe realizarse en un tiempo muy corto, ya que, si no, introducimos demasiado retardo en el sistema al ser un componente que se utilizará continuamente (al contrario que el generador, que sólo se utiliza para obtener el conjunto bruto de historias).

A continuación, nos disponemos a describir brevemente cada una de estas partes, de modo que se puedan tener referencias de la estructura de la aplicación para, más adelante, poder entrar en detalle con cada una.

3.2.1. Generador de historias

Este trabajo se basa en gran medida en el trabajo de fin de grado "Generación automática de historias basada en agentes software"

Cambio de paradigma

Durante el desarrollo del presente proyecto surgió la oportunidad de participar en el proyecto europeo WHIM (ver Marco de trabajo: WHIM (The What-If Machine), en la página 11), al cargo de Pablo Gervás, debido a que uno de sus componentes hacía uso de un generador de historias. Por ello, se pensó que, a fin de poder sacar conclusiones sobre el comportamiento global de nuestro sistema, sería interesante conocer el conjunto completo de historias que el sistema es capaz de generar, y para ello el enfoque multiagente era ineficiente. Puesto que en ese caso cada personaje tendría que realizar todas las acciones posibles para cada momento dado, un sistema multiagente habría consumido demasiados recursos (además, debemos tener en cuenta que inicialmente nuestra idea era crear historias muy grandes para después obtener las partes que son interesantes).

Por ello, decidimos estructurar nuestro generador de historias en forma de gramática formal, pasando a generar historias de menor longitud, pero en mucho mayor número. En este enfoque tenemos un conjunto de reglas que definirán nuestra historia y, a partir de la aplicación de unas u otras en cada momento, podremos generar historias diferentes.

Fortalezas y debilidades

El uso de gramáticas formales resulta conveniente para nuestro proyecto por varias razones. Por cómo están estructuradas las gramáticas formales, es muy fácil añadir nuevas posibilidades mediante la inserción de nuevas reglas, lo que hace nuestro sistema fácilmente ampliable. Aunque de esta forma perdemos mucha riqueza en cuanto a la interacción entre los personajes y el entorno, cada historia se genera en un plazo de tiempo muy corto, por lo que podemos generar todas las posibilidades en un tiempo asumible. Además, puesto que accedemos a los hechos acontecibles a través de la página web proporcionada por el grupo de investigación sobre generación de narrativa de la Universidad de Cambridge [1], las historias que podemos generar varían en función de los hechos que se proporcionen como respuesta a través del servicio web, lo que nos da gran cantidad de posibilidades.

Sin embargo, las gramáticas formales poseen una serie de inconvenientes que es importante tener en cuenta. Una de esas características es la explosión de estados. Dado que cada historia en particular sigue una rama de aplicación de reglas diferente, cada

[1] http://ccg02.doc.gold.ac.uk/eventchains3/entity_narrative/api/query/word2vec/gigaword-300-full-with-args-trans-adj/

nueva regla multiplica el número total de historias que somos capaces de generar, haciendo que el crecimiento sea exponencial. Es por esto que siempre tenemos un límite de posibilidades a partir del cual desbordamos el sistema. No obstante, si cada una de las reglas añadidas es lo suficientemente rica, no sólo aumentamos rápidamente el número de historias posibles, si no que añadimos mucha variedad a las historias generadas. Por tanto, es necesario mantener limitado el número de reglas que posee nuestra gramática.

Otra desventaja de esta forma de generar historias es que tenemos todo el contenido predefinido estructuralmente por el programador. Esto hace que las historias tengan una estructura muy concreta, lo cual es ventajoso en cierto sentido, ya que garantiza que las historias sean coherentes estructuralmente (dado que dicha estructura está definida por el programador). Sin embargo, mientras que en la generación basada en agentes los personajes tienen la capacidad de dirigir la historia realizando acciones de forma autónoma, en la generación basada en gramáticas los personajes son una pieza más de la historia, y no pueden aportar nada por sí mismos al desarrollo. De hecho, generalmente en este tipo de sistemas, los personajes suelen funcionar como "ladrillos de construcción" que se sustituyen inopinadamente, lo que puede dar lugar a historias incoherentes desde el punto de vista de la plausibilidad o la lógica. Aunque en el caso que nos ocupa este último problema no lo es tanto (ya que la idea es generar historias en respuesta a what-ifs que no tienen que ser totalmente lógicos necesariamente) puede suponer un problema en algunos casos, y es necesario tenerlo en cuenta.

Conclusión

En nuestro caso, resulta conveniente el uso de gramáticas gracias a que las historias que generamos tendrán una estructura común, debido a la naturaleza del proyecto (se recibe como archivo de entrada una descripción que contiene el contexto junto con los puntos principales que deben aparecer en la historia, divididos en planteamiento, nudo y desenlace). Además, esto implica que el contexto de las historias generadas puede cambiar por completo, por lo que el uso de una gramática capaz de adaptar los elementos de la historia con muy bajo coste computacional es muy favorable. Por otro lado, el uso de una gramática nos permite extraer el elemento aleatorio presente en el proyecto en el que nos basamos (qué personajes interactúan entre sí, qué acciones realizan, etc.), permitiendo conocer el conjunto completo de historias (lenguaje) que podemos crear, lo que es de mucha ayuda a la hora de realizar estudios sobre el programa.

3.2.2. Modelo de interés de una historia

Dado que las historias generadas utilizando gramáticas formales son combinaciones de los elementos de una historia disponibles para el sistema, es lógico pensar que habrá historias que no tengan ningún interés para el lector. Por ello, decidimos que era necesario aplicar algún tipo de filtro que nos dijera qué historias son las mejores, y se pensó que sería una buena opción aplicar técnicas de aprendizaje automático, a causa de la

dificultad actual para encontrar los elementos de interés en una historia. Además, aunque ya se han utilizado elementos de aprendizaje automático con anterioridad en el campo de la narrativa, no existe ningún generador que lo aplique a la hora de predecir el interés, por lo que es una línea de investigación muy interesante, ya que aborda directamente un problema que afecta a diferentes campos, siendo además uno de los principales problemas de la generación de historias. Una ventaja del uso de esta técnica en detrimento de otras es que es aplicable a cualquier dominio de historia, por lo que nos permite extraer conocimiento global sobre el interés en una historia.

Una vez entrenado, el modelo actúa como filtro de nuestro sistema, de forma que sólo se presenten al usuario las historias que más probablemente sean interesantes. Una vez entrenado, los resultados pueden utilizarse no sólo para mejorar el propio clasificador (realimentándolo con historias interesantes, de forma que cada vez encontremos atributos más fuertemente ligados al interés de la historia), sino para mejorar la gramática misma, de forma que el conjunto global de historias sea más interesante.

Para simplificar las cosas, el aprendizaje máquina se aplica a las historias generadas sin ningún tratamiento, es decir, directamente sobre el conjunto de sucesos que la componen. Por lo tanto, no tenemos en cuenta atributos que tengan que ver con el estilo del texto en lenguaje natural propiamente dicho, aunque consideramos que sería una línea de investigación interesante.

Fortalezas y debilidades

Hemos utilizado usuarios reales para valorar lo interesantes que son las historias que generamos. Con esto, obtenemos la clase (interesante o no interesante) de cada historia. Una vez hecho esto, debemos encontrar los parámetros (*features*) de clasificación óptimos (los más fuertemente ligados al interés) que, junto con el valor de la clase, servirán para entrenar un sistema de aprendizaje máquina basado en *Support Vector Machines*. Con el objetivo de generar el menor rechazo posible en los usuarios a la hora de valorar el interés de cada historia, decidimos traducir las historias a lenguaje natural mediante el uso de plantillas (ver Generación de lenguaje natural, en la página 47).

Por cómo está estructurada la generación del modelo de interés, no es necesario modificarlo en caso de ampliar la gramática que genera las historias. En caso de cambiar el paradigma del generador utilizado, sería necesario añadir y/o alterar las métricas (atributos) utilizadas para el análisis de las historias, aunque el proceso de generación del modelo no se vería alterado, por lo que el coste del cambio se ve sensiblemente reducido.

Sin embargo debemos tener cuidado, ya que hay que tener en cuenta que los resultados son totalmente dependientes de los atributos que utilicemos a la hora de clasificar. En otras palabras, el modelo clasificará de una forma más precisa si escogemos los atributos adecuados durante la fase de entrenamiento.

Conclusión

Dada la falta de consenso en relación a qué características hacen interesante una historia, el uso de esta técnica nos resulta muy favorable (ver Aprendizaje automático, en la página 15). Como hemos dicho, el aprendizaje automático es una rama de la inteligencia artificial que permite crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos (en nuestro caso, las historias que generamos). Dado que las historias son valoradas por personas reales, podemos decir que los atributos ideales a partir de los cuales el sistema clasifica de forma óptima coinciden con elementos que hacen que una historia tenga interés. Por tanto, es necesario entrenar el clasificador en función de múltiples conjuntos de atributos, hasta encontrar los que están más fuertemente relacionados con el nivel de interés obtenido.

Capítulo 4

Implementación

En este capítulo describimos en detalle cada uno de los componentes del sistema a nivel de código. Explicaremos las funcionalidades más importantes en cada caso, poniendo especial atención en las particularidades de su implementación y las razones por las que se hizo de esa forma.

4.1. Generador de historias

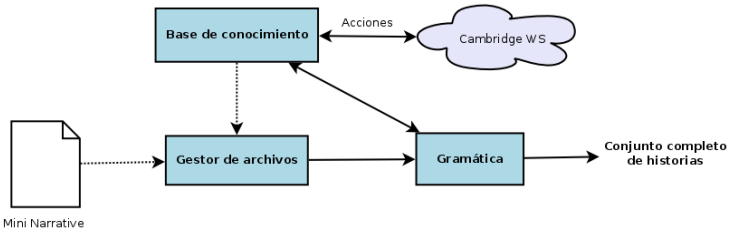
El generador de historias es el encargado de aplicar las reglas contenidas dentro de la gramática principal del sistema. Este componente (ver figura 4.1 en la página 40) consta de tres módulos propios, todos implementados utilizando Prolog, y un módulo externo, que se comunica con el sistema.

- Base de conocimiento: contiene los procedimientos prolog necesarios para la correcta generación de contenido, como frases, hechos o el intérprete del servicio web.
- Gramática: describe la estructura de las historias, eligiendo lo que ocurrirá en cada momento.
- Gestor de archivos: encargado de cargar la *mini narrative* (ver Mini Narrative, en la página 12) y la base de conocimiento en el sistema, de forma que la gramática pueda hacer uso de ellos. Contiene el método principal, encargado de lanzar la gramática y mostrar la lista de historias generadas.

Además, existe un módulo externo que se comunica con el sistema, encargado de proporcionar las acciones. Las acciones son posibles caminos que puede tomar un personaje en cada momento, y son cargadas dinámicamente utilizando un servicio web provisto por la universidad de Cambridge.

La herramienta utilizada para el manejo de Prolog fué SWI-Prolog, una implementación versátil del lenguaje de programación lógica Prolog. Permite la representación estructurada de documentos (como XML, JSON) y el intercambio de datos con otros

Figura 4.1: Componentes del generador de historias



paradigmas de programación. Para la implementación de este proyecto se utilizó la versión SWI-Prolog 7.2.3.

4.1.1. Archivos de entrada

Nuestro sistema posee un archivo de entrada que contiene lo que llamamos *Mini Narrative* (ver Mini Narrative, en la página 12). Con estos datos el sistema es capaz de generar la historia mediante la introducción de sucesos tras la aparición de los *narrative points*, creando así una historia en respuesta al *What-If*. Cabe destacar que cada parte de la historia (planteamiento, nudo y desenlace) tiene sus propios *narrative points*, que serán los únicos que se consuman (y añadan a la historia) durante dicha fase de la gramática.

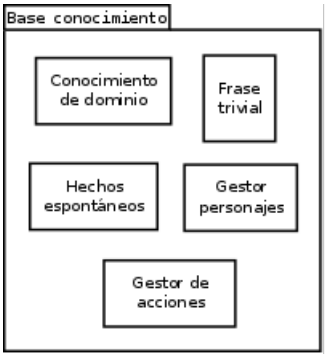
Tabla 4.1: Ejemplo de una mini narrative. Modificando los narrative points presentes alteramos la historia

```
# What if there was an old dog, who could no longer run, which
# he used to do for fun, so instead he learned
# how to ride a horse [for fun]?

#Narrative np1, id1, ch, isa(dog, animal), given, 0, bydefinition
np1, id2, ch, doesfor(dog, run, fun), inferredby(id5), +1, assumed
np1, id3, ch, -capableof(dog, ride_horse), inferredby(id6), 0, assumed

np2, id4, ch, hasproperty(dog, old), given, -1, assumed
np2, id5, ch, -capableof(dog, run), given, -1, assumed

np3, id6, ev, learnto(dog, ride_horse), given, +1, inferredby(id7)
np3, id7, ch, capableof(dog, ride_horse), causedby(id6), +1, inferredby(id8)
np3, id8, ch, doesfor(dog, ride_horse, fun), inferredby(id2), +1, assumed

#schemas
np1 :: setting
np2 :: conflict
np3 :: resolution
```

4.1.2. Base de conocimiento

Este componente es el que contiene el conocimiento del sistema. Con él, aportamos mayor riqueza a las historias generadas, añadiendo posibles frases o caminos que se pueden seguir. Contiene métodos para generar frases, presentar y eliminar personajes, hechos y métodos de comunicación con la web. Para mayor claridad, dividiremos sus funcionalidades en las siguientes categorías (ver figura 4.2 en la página 41): conocimiento de dominio, frase trivial, hechos espontáneos, gestor de personajes y gestor de acciones.

Figura 4.2: Funcionalidades presentes en la base de conocimiento



La gramática utiliza este componente cada vez que se quiere introducir algún suceso a la historia. Por ello, es un componente crítico del sistema, ya que la calidad de las historias generadas depende en gran medida de la complejidad del mismo. Por otro lado, gracias a que todo el conocimiento está contenido aquí, es muy fácil añadir nuevas posibilidades, ya que las modificaciones necesarias en la gramática son triviales.

Conocimiento de dominio

Consiste en una serie de hechos que el sistema necesita para tener un mínimo conocimiento del mundo. Están formados por un conjunto de hechos prolog con cinco parámetros (ver tabla 4.2 en la página 41): contexto, estado actual de la historia, clase (animal, actividad, palabra positiva, etc), elemento (perro, gato, correr, diversión, etc) y probabilidad. Su uso principal es a la hora de distinguir los personajes de las palabras corrientes, por lo que se utiliza durante la introducción de los mismos.

Tabla 4.2: Ejemplo de conocimiento de dominio

domain(_, _, animal, dog, 1.0).
domain(_, _, animal, cat, 1.0).
domain(_, _, animal, horse, 1.0).
domain(_, _, activity, run, 1.0).
domain(_, _, activity, ride_horse, 1.0).

Su uso principal es a la hora de distinguir los personajes de las palabras corrientes (ver tabla 4.3, en la página 42), por lo que se utiliza durante la introducción de los mismos. Para ello, obtenemos todos los animales presentes en el conocimiento de dominio para, después, comprobar si alguno de ellos aparece en el *narrative point* actual (ver Gestor de personajes, en la página 44).

Tabla 4.3: Búsqueda de personajes

```
/* Obtiene los personajes de los narrative points */

all_characters(Pers, NP) :- findall(P, character(P, NP), Pers).

character(Pers, [P | NP]) :-
   domain(_, _, animal, Pers, _), is_character(Pers, [P | NP]).

is_character(Pers, [P | NP]) :-
   (has_character(Pers, P), !) ; character(Pers, NP).

has_character(Pers, SNP) :-
   format(atom(A), -w", SNP), sub_string(A, _, _, _, Pers).
```

Frases triviales

Estas frases se generan durante la fase de introducción de la historia, y contienen un contexto para la misma (ver tabla 4.4, en la página 42). Son triviales ya que hablan del día y la hora, o el clima.

Tabla 4.4: Estructura de una frase trivial

```
frase_trivial(T) :- calendar_sentence(T).
frase_trivial(T) :- climate_sentence(T).
```

Las frases de calendario (ver tabla 4.5, en la página 43) hacen referencia al momento en el que se produce la historia. Contienen información acerca del día del año, la hora actual, o ambas. Todos los datos son obtenidos del sistema anfitrión en tiempo de ejecución.

Las frases de clima (ver tabla 4.6, en la página 43) poseen tres partes: estado del cielo, estado del viento y temperatura. Los datos consisten en hechos introducidos a mano en la base de conocimiento.

Hechos espontáneos

Llamamos hechos espontáneos (ver tabla 4.7 en la página 43) a aquellos que pueden suceder en cualquier momento de la historia, como que empiece a llover o que o que, en

Tabla 4.5: Frase trivial: calendario

```
calendar_sentence(F) :-
   get_time(T), stamp_date_time(T, D, local), calendar(D, F).

calendar(Date, [date(Yr, Mth, Day)]) :- mn_date(Date, Yr, Mth, Day).
calendar(Date, [time(Hr, Mn)]) :- mn_time(Date, Hr, Mn).
calendar(Date, [date(Yr, Mth, Day), time(Hr, Mn)]) :-

mn_date(Date, Yr, Mth, Day), mn_time(Date, Hr, Mn).
```

Tabla 4.6: Frase trivial: clima

```
climate_sentence(F) :- sky(S), wind(W), temp(T),
append(S, W, Int), append(Int, T, F).

sky([clear(sky)]).
sky([cloudy(sky)]).
sky([rainy(sky)]).

wind([windy(wind)]).
wind([-windy(wind)]).

temp([cold(temperature)]).
temp([hot(temperature)]).
temp([warm(temperature)]).
temp([nice(temperature)]).
```

una película de disney, los personajes empiecen a cantar una canción. Están estructurados en forma de hechos prolog con tres parámetros: contexto, acción posible y probabilidad.

Tabla 4.7: Algunos hechos espontáneos

```
spontaneous_fact(_, start(rain), 0.5).
spontaneous_fact(_, start(fight), 0.4).
spontaneous_fact(disney, sing(people), 0.3).
```

Por otro lado, cada una de estas acciones acarrean consecuencias (ver tabla 4.8 en la página 44) para todos los personajes presentes en la historia. Por ejemplo, cuando empieza a llover los personajes pueden mojarse o sacar un paraguas. Esto dota a las historias de un mayor realismo, en el sentido de que las hace más cercanas al usuario. Para ello, tenemos otra serie de hechos prolog correspondientes a las consecuencias, con tres parámetros: el hecho primario, el personaje y la consecuencia producida. Decidimos que estos hechos estuvieran puestos directamente en el código debido a que la lentitud del

servicio que proporciona las acciones (ver más abajo) hacía que los tiempos de ejecución subieran considerablemente.

Tabla 4.8: Consecuencias para los hechos

spontaneous_effect(start(fight), P, get(P, angry)). spontaneous_effect(start(fight), P, get(P, scared)). spontaneous_effect(start(fight), P, preparefor(P, fight)).

Gestor de personajes

El generador lleva un control de los personajes que aparecen durante la historia. De esta forma, podemos actuar con conocimiento de quién aparece. Esto nos brinda la posibilidad de relacionar los nuevos personajes con los antiguos, de manera que nunca se añaden personajes a la historia de forma aleatoria. Así, conseguimos que la adición de personajes sea mucho más natural, parecida a como lo hacemos en la realidad.

Durante la fase de planteamiento de la historia se realiza la introducción de los protagonistas (ver figura 4.3). Consideramos que los personajes que aparecen en los narrative points de la introducción corresponden a los protagonistas de la misma. Estos personajes no tienen relación entre sí, si no que sirven como punto de apoyo a la hora de presentar personajes más adelante. En este caso, el mecanismo elegido es el aserto en el conocimiento del sistema de hechos que lo corroboren, de forma que la consulta de los mismos es muy eficiente.

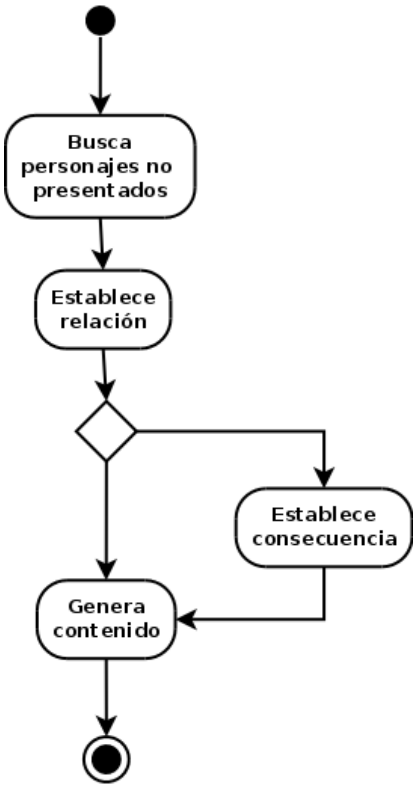
Figura 4.3: Proceso de introducción de los protagonistas

Aquellos personajes presentados en otras fases se corresponden a personajes secundarios (ver figura 4.4, en la página 45). Estos personajes deben guardar una relación con los principales, de forma que siempre está justificada su presencia. Por ejemplo, el personaje secundario puede ser amigo o primo de uno de los protagonistas. Además, se han añadido consecuencias a esas relaciones (por ejemplo, si un personaje es hermano de otro, eso implica que ese personaje quiere al protagonista). Sin embargo, hemos decidido que no hacer necesaria la inclusión de consecuencias a las relaciones. A la hora de la presentación, en este caso se decidió que los personajes secundarios no fuesen añadidos a la base de hechos del sistema, ya que producía problemas durante el proceso de *backtracking* de prolog. En su lugar, se optó por obtener los personajes secundarios que se han introducido mediante la búsqueda en el estado de entrada de la historia. No se introdujeron cambios en cuanto a los personajes principales debido a que resulta un método mucho más eficiente y a que no daba problemas.

Tabla 4.9: Ejemplo de relaciones

```
relation_fact(P1, P2, brother(P1, P2)).

relation_effect(brother(P1, P2), love(P1, P2)).
relation_effect(brother(P1, P2), hate(P1, P2)).
```

Figura 4.4: Proceso de introducción de los personajes secundarios



Para simplificar, se tomó la decisión de que los personajes fuesen aquellos elementos del dominio etiquetados como animales (en nuestro caso *dog, cat* y *horse*). Por esta razón, decidimos situar nuestras historias en el contexto de los cuentos de Disney. Sin embargo, sería interesante permitir que el tipo de protagonista dependa del contexto de la historia (ver Trabajo futuro, en la página 66).

Gestor de acciones (servicio web)

En este apartado englobamos los métodos utilizados para hacer uso del servicio web del grupo de investigación sobre generación de narrativa de la Universidad de Cambridge. Contiene un parser de acciones, encargado de obtener los parámetros de la acción que se le pasa como argumento para realizar la petición al servidor. Estos argumentos se corresponden con la estructura de la frase, como sujeto, objeto o complemento directo en caso de que lo hubiera. Una vez obtenidos, se le envían al servicio web, que los usa para generar nuevas acciones que guarden relación con la misma.

El servicio web tiene un funcionamiento muy sencillo. Puede recibir varias consultas, y nos devuelve una lista de acciones posibles (ver tabla 4.10 en la página 47) en relación con cada una de ellas, ordenadas decrecientemente en función de su probabilidad. Por ejemplo, en el caso de pedir acciones posibles para “hombre come pastel” y “dinosaurio conduce coche”, nos devolvería cosas como "X bebe licor.º "X seca nueces"para el primero y "X resbala en un bote.º "X aterriza en un colchón", todo ello en formato JSON. En nuestro caso se decidió que la acción pasada en la petición sería una de las acciones contenidas en la base de conocimiento del sistema.

Una vez recibida la respuesta, es necesario pasar del archivo JSON a una acción interpretable por el sistema. De ello se encarga otro parser, que obtiene los campos del documento JSON que nos interesan (en nuestro caso la acción en sí misma y su probabilidad asociada) y sustituye las variables que contienen por elementos presentes en la historia (por ejemplo, pasando de “X eats _____” a “Dog_eats_cat”).

Acciones

Gracias al servicio web que nos proporciona la Universidad de Cambridge, el sistema hace uso de una base de datos on line para acceder a las diferentes acciones posibles por parte de un personaje de forma dinámica. De esta forma, tenemos acceso a una base de conocimiento mucho mayor que la que podríamos generar, además de ganar en riqueza (acciones mucho más variadas).

Este tipo de acciones sólo se utilizan en la fase de desarrollo de la historia. Esto es debido a que el acceso al servicio web es bastante lento, lo que lo convierte en un cuello de botella para la aplicación. Por esta razón se ha decidido que cada historia contenga una sola acción, aunque sería interesante añadir más. Sin embargo, a pesar de esta desventaja consideramos que su uso es muy interesante ya que añade múltiples posibilidades a las

Tabla 4.10: Ejemplo del JSON recibido como respuesta del servicio web

{"predicate":"munch:subj","score":0.9952253726613094,ïobject":,"subject":"X",
"verb_lemma":"munch","preposition":,"verb":"munch","type":"normal",
.event":"X munches treat",.object":"treat"},

{"predicate":"taste:subj","score":0.9940865011540836,ïobject":,"subject":"X",
"verb_lemma":"taste","preposition":,"verb":"taste","type":"normal",
.event":"X tastes liquor",.object":"liquor"},

{"predicate":roast:subj","score":0.992870163796369,ïobject":,"subject":"X",
"verb_lemma":roast","preposition":,"verb":roasted","type":"normal",
.event":"X roasts fish",.object":"fish"},

{"predicate":"gulp:subj","score":0.9920020089907626,ïobject":"bottle","subject":"X",
"verb_lemma":"gulp","preposition":"from","verb":"gulped","type":"normal",
.event":"X gulps water from bottle",.object":"water"}

historias del sistema. Además, tiene la ventaja de que se obtienen de manera dinámica, por lo que, añadiendo nuevas acciones a la base del conocimiento (aquellas que utilizamos para hacer las peticiones al servicio web), las acciones recibidas como respuesta serán muy diferentes, con lo que las posibilidades aumentan exponencialmente.

Sin embargo, el hecho de que las acciones no sean conocidas en tiempo de compilación supone un problema a la hora de generar automáticamente las *features* de las historias (ver Filtro de interés de las historias, en la página 51) ya que no es fácil, por ejemplo, discernir si una acción es divertida, dramática, alegre o violenta.

Durante la ejecución de la gramática, el sistema accede a las distintas acciones posibles, generando una historia (con distinta probabilidad) por cada una de ellas.

4.1.3. Generación de lenguaje natural

Por sencillez, decidimos afrontar la generación de texto en lenguaje natural mediante el uso de plantillas predefinidas, aunque en un principio nos planteamos el uso de un generador de lenguaje natural externo.

Estas plantillas (ver tabla 4.11, en la página 48) se encargan de transformar cada uno de los hechos de la historia (que están bien estructurados) en texto, obteniendo la acción concreta junto con los actores (para mayor detalle, ver apéndice B, en la página 73).

Tabla 4.11: Ejemplo de plantillas de texto

```
textualize([isa, X, Y], N) :-
  atom_concat('The ', X, Int),
  atom_concat(Int, ' was an ', In), atom_concat(In, Y, N), !.
textualize([doesfor, X, Y, Z], N) :- atom_concat('The ', X, In),
  atom_concat(In, ' ', Inner), (translate(do, Y, T), atom_concat(Inner, T, In2) ;
  atom_concat(Inner, Y, Int),
  atom_concat(Int, 's', In2)), atom_concat(In2, ' for ', In3),
  atom_concat(In3, Z, N), !.
textualize([capableof, X, ride_horse], N) :- atom_concat('The ', X, Int),
  atom_concat(Int, ' was capable of ', In), atom_concat(In, 'riding a horse', N), !.

translate(learn, ride_horse, 'ride a horse').
translate(capable, ride_horse, 'riding a horse').
translate(do, ride_horse, 'rode a horse').
translate(do, run, 'ran').
translate(capable, run, 'running').
```

4.1.4. Gramática

La gramática del sistema (ver apéndice A, en la página 71) está implementada utilizando la utilidad de Prolog dedicada a ello (DCG, *Definite Clause Grammar*). En el caso que nos ocupa, consideramos que las historias tienen tres partes: planteamiento, nudo y desenlace (*beginning, development y outcome*). Para mantener la coherencia interna, cada una de las partes debe poseer información sobre lo que ha ocurrido anteriormente, por lo que la salida de cada parte se utiliza como entrada de la siguiente (ver tabla 4.12 en la página 49). Dicha información incluye:

- Contexto (WV): es el tipo de historia que se generará (por ejemplo, disney, western, sci-fi ...).
- Mini Narrative (MN): contiene la lista de narrative points que nos queda por consumir. Se utilizan para saber los elementos de la historia que nos quedan por presentar.
- MN Output (MNO): contiene los narrative points que utilizarán las siguientes partes de la historia.
- Estado (St): contiene la lista de consecuencias de las acciones que se han producido durante la historia. Se utilizan para conocer los hechos acontecidos en la historia, de forma que podamos tomar decisiones en consecuencia más adelante.
- St Output (StO): representa el estado al final de la ejecución de esta parte de la historia.

- Probabilidad (C): porcentaje de plausibilidad de la historia generada hasta el momento. La probabilidad final será el producto de la probabilidad de cada parte.

Tabla 4.12: Estructura de la historia

```
story(WV, MN, MNO, St, StOut, C) –>
        restart,
        beginning(WV, MN, MNB, St, StB, C1),
        development(WV, MNB, MND, StB, StD, C2),
        outcome(WV, MND, MNO, StD, StOut, C3),
        { C is C1 * C2 * C3 }.
```

Dado que para la presentación de los personajes principales nos decantamos por la opción de añadirlos al conocimiento del sistema, es necesario descartar todos los personajes presentados antes de generar una nueva historia. De esta forma, al realizar el proceso de backtrack, cada historia presenta sus personajes, y los añade al conocimiento del sistema cada vez. Esto se realiza durante la fase de restart. En todas las fases de la historia se presentan los narrative points correspondientes a esa fase, y además se añade algo de contenido a la historia, que depende de la fase concreta en la que nos encontremos. Esto se realiza mediante llamadas a la base de conocimiento del sistema, que contiene los métodos necesarios para generar contenido.

Planteamiento

En esta fase se realiza la presentación de los personajes principales de la historia. Consideraremos personajes principales a aquellos que aparezcan en los narrative points de la introducción, y la historia girará en torno a ellos (el resto de personajes aparecerán en la historia en relación con los protagonistas). Los personajes introducidos se añaden al estado de salida de la historia (dado que en este caso llevamos la cuenta de los personajes principales mediante la adición de conocimiento, no es necesario añadirlo al estado, aunque se ha añadido debido a su similitud con el caso de los personajes secundarios).

Opcionalmente existe la posibilidad de añadir una frase trivial al comienzo de la historia. Estas frases nos situarán en un contexto, dándonos fecha, hora y tiempo en el lugar de la historia.

Nudo

Es la fase encargada de generar el contenido principal de la historia. En este punto, los personajes principales ya han sido introducidos, y consideramos personajes secundarios a aquellos que aparezcan en esta fase y las posteriores. A la hora de presentarlos, se les asignará una relación con algún personaje principal, para añadir coherencia. Esta relación puede producir una consecuencia (ver tabla 4.13 en la página 50) en la historia o no.

Figura 4.5: Diagrama de flujo de la fase de planteamiento

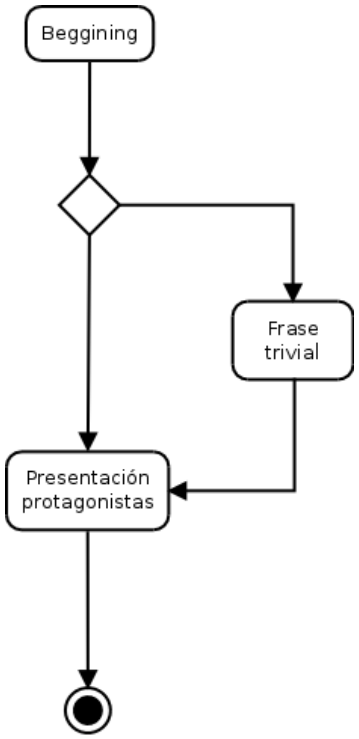


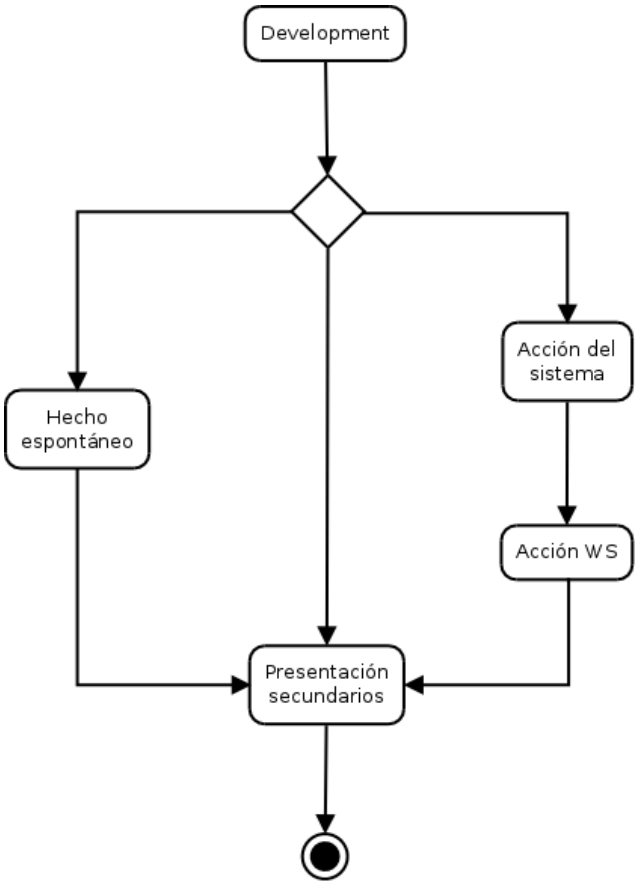
Tabla 4.13: Ejemplo de relación y su consecuencia

enemy(P1, P2) ->hate(P1, P2)

Además de esto, en esta fase pueden producirse hechos espontáneos. Estos hechos podrían ocurrir en cualquier momento de la historia, aunque por simplicidad se ha decidido que sólo puedan ocurrir durante el desarrollo. Los hechos espontáneos afectan a todos los personajes de la historia, produciendo un efecto (p.e. comienza a llover, el personaje A saca un paraguas, el personaje B se moja).

Por último, en esta fase se realiza la consulta al servicio web de la Universidad de Cambridge para obtener posibles acciones. En primer lugar, se aplica una acción escogida entre un pequeño conjunto disponible en la base de conocimiento local. Una vez hecho esto, el sistema contacta con el servicio web de la Universidad de Cambridge para encontrar las acciones que sucederán después de la misma, y escoge una (formato JSON), quedándose con el texto de la acción (el campo "event", en el que se sustituyen sujeto y objeto por elementos presentes en la historia) y su probabilidad de ocurrencia.

Figura 4.6: Diagrama de flujo de la fase de nudo



Desenlace

Por simplicidad, esta fase se encarga simplemente de presentar los narrative points finales.

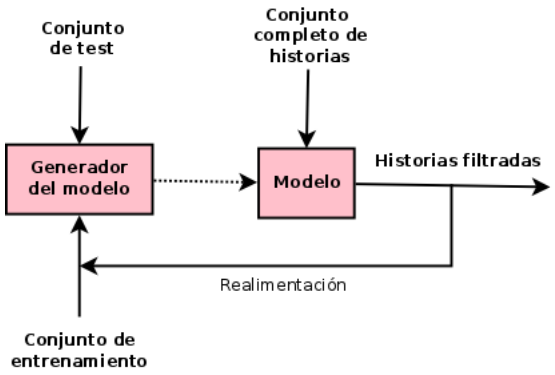
4.2. Filtro de interés de las historias

Mediante el uso de la herramienta Weka (ver Herramientas para aprendizaje automático, en la página 19), hemos entrenado un modelo de historia interesante en función de diferentes paradigmas. Dado que la creatividad y todo lo que la rodea está formada por conceptos muy vagos, es muy difícil determinar a simple vista qué elementos son los que más inciden en el interés de una historia. Por ejemplo, podríamos pensar que los hechos que afecten a más personajes serán los más interesantes, pero puede no tener ningún interés en la historia el hecho de que suba el precio del pan (afecta a todos).

Para mejorar los resultados se han generado varios modelos diferentes, utilizando diferentes funciones para el *kernel* del SVM (ver Support Vector Machines, en la página 17) en cada uno, utilizando la metodología típica de aprendizaje máquina. Extrajimos un subconjunto de 200 historias (distribuidas aleatoriamente) del conjunto completo, y generamos dos subconjuntos:

- Conjunto de entrenamiento: es el mayor de los dos, y contiene una serie de historias con atributos y valor (interesante/no interesante), que sirven al sistema para obtener los parámetros más relevantes.
- Conjunto de test: se utiliza para comprobar la eficacia del modelo, pasándole cada una de las historias sin su valor y comprobando el resultado obtenido con la etiqueta asignada previamente.

Figura 4.7: Estructura del filtro de las historias



Gracias a la herramienta Weka podemos comprobar, además del índice de acierto en la clasificación del conjunto de test, el peso específico de cada argumento para el conjunto de entrenamiento especificado. Mediante el uso de diferentes conjuntos de entrenamiento y test, podemos calcular una media del peso de cada atributo, haciéndonos así una idea de su importancia para el interés. Lo mismo ocurre con diferentes conjuntos de atributos. Al incluir nuevos atributos junto a aquellos que ya hemos utilizado, su peso variará para esos conjuntos, por lo que también debemos tenerlo en cuenta a la hora de calcular el peso medio.

Es importante notar que el cálculo del valor de dichos atributos debe ser automático, ya que nuestro sistema genera un conjunto de historias demasiado amplio como para hacerlo a mano. Así, podremos añadir nuevos atributos simplemente codificando una función que los calcule. Debido a la forma en que se estructuran las historias (listas de hechos) hemos decidido utilizar Prolog para implementar dichas funciones. Dado que las historias son listas de sucesos, es muy fácil moverse a través de ellas gracias al encaje de patrones que nos ofrece el lenguaje, por lo que su implementación es trivial, además de eficiente. Para ello, se ha implementado un archivo que contiene todos los métodos necesarios para el cálculo de los atributos de cada historia, además de su lectura y

escritura para posterior uso de la herramienta Weka, junto con las plantillas de texto necesarias para su traducción a lenguaje natural.

4.2.1. Métricas

El sistema incluye un archivo destinado al cálculo automático de los *features* que mostraremos a continuación (ver tabla 4.15, en la página 55), así como a la creación de los archivos necesarios (formato arff), correspondientes a los conjuntos de entrenamiento y test que el sistema de aprendizaje máquina utiliza. Debido a la necesidad de utilizar usuarios reales para la clasificación, decidimos incluir generación de texto en lenguaje natural basada en plantillas sencillas (ver Generación de lenguaje natural, en la página 47). Por ello, el texto generado es tosco, y posee faltas de ortografía en algunos puntos, pero hace las historias mucho más legibles y atractivas para los usuarios.

A continuación, haremos una lista de las métricas utilizadas como *features* de las historias en nuestro proyecto (ver tabla 4.14 en la página 54). Algunas de ellas (marcadas con * en la tabla) son resultado del trabajo realizado con métricas del proyecto WHIM, y las hemos adaptado para aplicarlas a historias ya generadas (las métricas originales estaban pensadas para las MN). Además de los *features* aquí descritos, se decidió utilizar a su vez una técnica conocida como "bag of words". En ella, se añade un atributo por cada palabra que se encuentre en el conjunto de entrenamiento, asignándole un 1 si aparece en la historia y un 0 en caso contrario. Esto puede servirnos para encontrar palabras (o combinaciones de ellas) que aporten interés a una historia.

Debido a la manera en que Prolog gestiona la lectura de archivos, nos interesa hacer que cada historia sea un predicado del lenguaje. Esto es debido a que, de esta forma, el cálculo de las métricas es mucho más sencillo, ya que tratamos con predicados del lenguaje en lugar de cadenas de texto, pudiendo así aprovechar todas las ventajas que el propio lenguaje nos ofrece. Para ello, leemos las historias como cadenas de texto plano, y las añadimos una vez convertidas a un archivo temporal, a partir del cual podemos leer los predicados correctamente. Gracias a que las historias han sido generadas usando el propio lenguaje, el proceso de conversión se reduce a añadir un punto al final de cada línea. Hecho esto, el programa calcula las *features* de cada historia automáticamente, y las escribe en un archivo con extensión arff, de forma que sirven como entrada para los modelos de predicción proporcionados por Weka. Gracias a esto, la generación de conjuntos de entrenamiento y prueba es muy sencilla y aplicable a conjuntos de datos (historias) muy grandes.

Aunque la lectura de predicados propios del lenguaje es muy sencilla en Prolog, la lectura de líneas de texto plano requiere mayor tratamiento (ver figura 4.8, en la página 56). En nuestro caso, en un principio utilizamos una gramática para definir la estructura de una línea para, posteriormente, proporcionar dicha gramática a la función de lectura de archivos propia del lenguaje. Sin embargo, esta forma de lectura desborda el sistema para conjuntos muy grandes de historias, por lo que finalmente hemos optado por una lectura línea a línea convencional.

Tabla 4.14: Métricas utilizadas como *features* en las historias

Métrica	**Descripción**
total_characters*	Número total de personajes
main_characters*	Número de protagonistas
secondary_characters	Número de secundarios
ratio_characters*	Relación entre el número de personajes protagonistas y secundarios
length*	Longitud de la historia (número de sucesos)
total_facts	Número de hechos (todo aquello que no son acciones)
ratio_characters_facts	Relación entre el número de personajes y el número de hechos
evolution*	Cantidad de conocimiento adquirido (número de learn-to)
main_character_actions	Número de acciones (events) que realiza el protagonista
total_actions	Número de acciones totales en la historia
ratio_actions	Relación entre el número de acciones del protagonista y el número total en la historia
love_level	Cantidad de amor presente en las relaciones (brother, friend...)
hate_level	Cantidad de odio en las relaciones (enemy, start fight)
drama_level	Cantidad de drama presente (start fight, start rain...)
fun_level	Nivel de diversión de la historia (get wet, get scared)
plausibility	Probabilidad de que suceda la historia
has_date	Indica si la historia tiene información sobre la fecha
has_time	Indica si la historia tiene información sobre la hora
has_weather	Indica si la historia tiene información sobre el tiempo
Bag Of Words**	Esta técnica añade una nueva métrica (*feature*) por cada palabra existente en las historias, y le asigna un 1 en caso de aparecer y un 0 en caso contrario

4.2.2. Clasificación según el interés

Una vez calculadas las *features*, utilizamos la herramienta Weka para entrenar el sistema. La herramienta nos permite elegir el conjunto de *features* con el que queremos entrenar, y nos proporciona filtros para el bag of words. Como una primera aproximación, utilizamos un conjunto de 182 historias etiquetadas a mano por el programador, dividiéndolo en dos conjuntos (entrenamiento y test) y utilizando diferentes modelos para la clasificación. Una vez entrenados, escogemos entre uno u otro basándonos en el índice de acierto que tiene. Para ello, hemos implementado un programa en Java (usando la librería de Weka) que entrena y testea diferentes modelos. Para nuestro caso, se han utilizado *Support Vector Machines* y Random Forest. Una vez comprobado que el

Tabla 4.15: Ejemplo de conversión de una historia en un conjunto de *features*, sin incluir el uso de bag of words (el orden que siguen es el mismo que en la tabla anterior).

It was Thursday, the 21 of April, 2016, and the time was 10:3. The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. The cat learned to ride a horse. The event horse_poops occurred. The cat was a secondary character. The cat was dogs enemy, so the dog hated the cat. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.
3, 2, 1, 0.6666666666666666, 17, 4, 0.75, 2, 3, 5, 0.6, 0, 1, 1, 0, 0.17140300571918488, 1, 1, 0

modelo que mejor resultado ofrecía era el de *Support Vector Machines*, nos propusimos recabar opiniones de usuarios reales. Utilizando la herramienta Google Forms, realizamos 10 cuestionarios con 20 historias cada uno (escogidas a partir del conjunto completo de historias, siguiendo una distribución aleatoria para evitar que sean demasiado similares entre sí), para saber cuáles eran interesantes y cuáles no. Hemos intentado que cada cuestionario tenga al menos 3 valoraciones para asegurar la correctitud de los resultados. Gracias a ello hemos podido generar conjuntos de entrenamiento y test basados en opiniones reales y, por tanto, un modelo basado en la realidad.

Una vez tenemos el modelo, basta con calcular las *features* para el conjunto completo de historias del sistema, y utilizarlo como entrada para el clasificador, que nos dirá si la instancia presentada en cada momento es interesante o no (en relación al modelo que hemos calculado).

Figura 4.8: Proceso de lectura de las historias

Capítulo 5

Resultados

En este capítulo podemos ver los resultados obtenidos al finalizar el proyecto, así como los objetivos que se cumplieron. Además, discutimos acerca de las fortalezas y debilidades de nuestro sistema frente a otros.

5.1. Resultados obtenidos

Hemos desarrollado un prototipo simple que utiliza todos los componentes descritos en las anteriores capítulos para generar un conjunto completo de soluciones, con un total de 40.460 historias cortas. Estas historias poseen una estructura común, pero son diferentes, basándonos en pequeñas variaciones de los elementos que en ellas aparecen, como los personajes o las acciones que se realizan. El sistema posee un modelo de interés generado a partir de valoraciones de usuarios reales, y es capaz de filtrar todas estas historias, dejando para el usuario final aquellas que más probablemente sean interesantes. El generador de historias es capaz de cambiar el contexto de las historias generadas, simplemente cambiando la *mini narrative* utilizada como archivo de entrada. Por último, el modelo generado no es capaz de mejorarse a sí mismo de forma automática, aunque es sencillo mejorarlo de forma manual añadiendo las historias mal clasificadas al conjunto de entrenamiento (con su nueva clase).

Para la generación del texto de la historia, utilizamos plantillas sencillas de lenguaje natural. De esta forma, las hacemos más legibles para el usuario, lo cual resultó especialmente útil durante la fase de recopilación de valoraciones de las historias.

En el tabla 5.1 podemos ver un ejemplo de historia (en forma de conjunto de hechos) generada por el sistema, y el resultado de su traducción mediante plantillas. En las tablas 5.2, 5.3, 5.4, 5.5 y 5.6 podemos ver algunos ejemplos de historias (ya traducidas) con la predicción del sistema.

El sistema obtuvo un porcentaje de acierto base del 42.564 % al clasificar las instancias de un conjunto de test generado por el programador, usando *Support Vector Machines*. Este porcentaje mejorará a medida que se interactúe con el sistema, añadiendo las instancias cuya valoración creemos que es errónea al conjunto de entrenamiento.

Tabla 5.1: Historia y traducción a lenguaje natural

[[date, 2016, 4, 21], [isa, dog, animal], [doesfor, dog, run, fun], [not, [capableof, dog, ride_horse]], [main_character, dog], [main_character, horse], [hasproperty, dog, old], [not, [capableof, dog, run]], [learnto, cat, ride_horse], [event, horse_euthanizes_dog], [secondary_character, cat], [parent, horse, cat], [love, horse, cat], [learnto, dog, ride_horse], [capableof, dog, ride_horse], [doesfor, dog, ride_horse, fun]], 0.18536169826984406
It was Thursday, the 21 of April, 2016. The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. The cat learned to ride a horse. The event horse_euthanizes_dog occurred. The cat was a secondary character. The cat was horses parent, so the horse loved the cat. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.

Tabla 5.2: Historia y su valoración 1

The sky was clear, the wind was not windy and the temperature was hot. The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. The cat learned to run. The event cat_doubles_off_horse occurred. The cat was a secondary character. The cat was dogs brother, so the dog loved the cat. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.
Instancia: 13515 Interesante

Tabla 5.3: Historia y su valoración 2

The sky was clear, the wind was windy and the temperature was cold. The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. The cat learned to ride a horse. The event dog_euthanizes_horse occurred. The cat was a secondary character. The cat was dogs friend. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.
Instancia: 6661 Interesante

Posteriormente, se realizaron 15 formularios con 20 historias cada uno mediante la herramienta Google Forms, para obtener el interés de usuarios reales en las historias que generamos, obteniendo entre 3 y 5 respuestas para cada test. Estos datos fueron utilizados para generar nuevos conjuntos de entrenamiento y test. Con estos nuevos conjuntos,

Tabla 5.4: Historia y su valoración 3

The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. It stareds to rain. The dog used umbrella. The horse used umbrella. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.
Instancia: 2 No interesante

Tabla 5.5: Historia y su valoración 4

The sky was rainy, the wind was not windy and the temperature was hot. The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. The cat learned to run. The event horse_gives_up_dog occurred. The cat was a secondary character. The cat was dogs enemy. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.
Instancia: 36899 Interesante

Tabla 5.6: Historia y su valoración 5

The sky was cloudy, the wind was windy and the temperature was warm. The dog was an animal, and he ran for fun. The dog was not capable of riding a horse. The dog and the horse were the main characters. The dog was old, so he was not capable of running. The cat learned to run. The event cat_gives_up_cat_to_dog occurred. The cat was a secondary character. The cat was horses friend, so the horse hanged out with the cat. The dog learned to ride a horse. The dog was capable of riding a horse, so, instead, the dog rode a horse for fun. The end.
Instancia: 20801 Interesante

se realizaron pruebas con los diferentes *kernels* disponibles en SVM (ver tabla 2.1, en la página 19), obteniendo un porcentaje de acierto final del 73.3333 % utilizando un *kernel* polinomial (el que mejor resultado daba). Este porcentaje nos indica el nivel de certeza con el cual clasificamos cada historia, ya sea interesante o no. En otras palabra, tenemos un 73.33 % de posibilidades de haber clasificado cada historia de manera correcta.

Una de las ventajas de nuestro sistema es su alto grado de modularidad. Esto es muy útil a la hora de añadir nuevas posibilidades al código, ya que podemos aumentar de forma significativa el contenido y la cantidad de historias realizando cambios mínimos. Además, esta característica nos brinda la posibilidad de cambiar los componentes del sistema individualmente. Por ejemplo, podríamos cambiar el generador de historias por

otro basado en agentes inteligentes sin que el sistema se viera afectado, siempre y cuando el nuevo generador mantuviese la estructura de las historias (en caso contrario tendríamos que entrenar un nuevo modelo, aunque no sería necesario cambiar el componente propiamente dicho).

5.2. Discusión

Para la realización de este proyecto nos hemos basado en algunas ideas ya exploradas en anteriores trabajos. Dado que nuestro trabajo utiliza gramáticas para la generación, nuestras mayores influencias han sido Novel Writer y Joseph. Aunque Talespin también realiza una aproximación utilizando reglas, estas se basan en el cumplimiento de objetivos, mientras que nuestro generador va "encajando piezas" utilizando la gramática como base.

Al igual que Joseph, nuestro sistema es altamente modular, lo que es una ventaja para el cambio del generador de historias (trabajo futuro) aunque, de no respetar la estructura de las historias, sería necesario generar un nuevo modelo. Como Joseph, hemos decidido dividir nuestras historias en planteamiento, nudo y desenlace debido a la naturaleza de nuestro proyecto (WHIM). Sin embargo, aunque podríamos incluir un módulo para la generación de lenguaje natural en un futuro, en nuestro caso era suficiente con generación basada en plantillas, ya que nuestro objetivo principal era mejorar la legibilidad para las valoraciones, por lo que es mucho más tosca que en el caso de Joseph.

Al contrario que Talespin o Virtual Storyteller, nuestro sistema no utiliza un mapa físico en el que se sitúen los personajes. En nuestro caso, de la misma forma que en Novel Writer (y en la mayoría de los generadores basados en gramáticas) nuestros personajes son un elemento más de la historia, y se sitúan de la misma forma que el resto de componentes, como acciones o hechos espontáneos, por lo que no tienen por ningún control sobre lo que ocurre en la historia por sí mismos. Además nuestros personajes no tienen ningún tipo de caracterización, en el sentido de que no se modelan sus sentimientos u objetivos. No obstante, debemos notar que esto no formaba parte de los objetivos marcados al comienzo del proyecto, por lo que queda fuera de su alcance.

Aunque algunos de los generadores que se han presentado como trabajos previos incluyen medidas de calidad propias de la aplicación (como en Talespin, donde una historia es satisfactoria si se ha cumplido el objetivo inicial del protagonista), resulta interesante notar que no existen sistemas de generación de historias que intenten encontrar los elementos interesantes de la historia de una forma más general. Utilizando métricas, el sistema filtra el conjunto total de sus historias, por lo que el resultado final tiene una mínima garantía de calidad, que depende directamente del porcentaje de acierto del modelo generado. Del mismo modo, aunque las historias que generamos son incluso más simples que las que generaban sistemas como Novel Writer, posee la ventaja de que es capaz de cambiar completamente el tipo de historias que genera en función de un archivo de entrada muy sencillo proporcionado, bien por WHIM o un usuario, mientras que en el

resto de generadores expuestos el cambio resulta más complejo de realizar y tiene mucha menor repercusión en el resultado final.

También resulta interesante notar que, aunque existen generadores que utilizan usuarios reales a la hora de la generación (generadores interactivos), no existe ningún sistema que utilice usuarios reales a la hora de valorar la calidad de los resultados.

Por último, aunque nuestro porcentaje de acierto es del 73.33 % debemos notar que es un resultado sólo válido para el tipo de historias que generamos actualmente, de modo que cambiando el generador de historias es muy probable que el porcentaje descienda considerablemente. Sin embargo, es importante destacar que este porcentaje se obtiene mediante el cálculo automático de métricas, por lo que, aumentando el número de ellas (ver Trabajo futuro) podríamos aumentar de nuevo dicho porcentaje para otros tipos de historia.

Capítulo 6

Conclusiones

En este capítulo exponemos las conclusiones a las que llegamos a partir de los resultados obtenidos, así como las diferentes líneas de trabajo que se nos ocurren y que no podemos continuar por falta de tiempo.

6.1. Conclussions

While we are not capable of modelling the human brain completely, we will not be able to reproduce how it works. This is specially critic in fields like creativity, in which we are not sure about how or why we act as we do.

Nevertheless, it is possible for us to generate bounded prototypes that help us comprehend some of the processes that take place in our brain. With our present knowledge, we are able to teach a computer to make up things, being in our case stories. The fact that creativity is such a vague concept gives us multiple research branches only regarding storytelling, such as feeling modeling, relations, interactions between characters and environment, etc.

Having worked in this field before, we have checked the advantages and disadvantages of our new approach relating to the previously studied (based on intelligent agents).

In relation to story generation, we consider agent based generation a much more interesting approach, as, in it, the characters build up the story themselves, in relation with their interactions and own goals, making stories much more natural. However, in our case it was necessary to know the complete set of possible solutions, making this approach inefficient. Besides, a great disadvantage compared to formal grammar based generation is the time it takes to make up a story. Agent's interaction is a complex computational process, while in grammars we only have to generate the complete solution tree. This tree is fixed, unlike in the agent based generation, as they use the information obtained (own goals, common goals, environment state, other agent's state, etc) for the generation.

For this reason, we consider the machine learning block as the most interesting of the system. Academically, the possibility of shedding some light over a field in which there have been several discussions (interesting elements of a story) without reaching a

consensus is highly motivating. In other way, the machine learning field is experimenting a peak nowadays, and it is called to have increasing importance thanks to new technologies, such as Big Data. We have checked the great amount of possibilities and algorithms that this field brings us, and the impact they have on the final results.

Regarding the features to pick in a data set, we have realized that its correct choice can be a research field in itself, as nowadays it is almost a craft work. In fact, it is noteworthy the fact that is a differentiating factor when choosing a data scientist in a company.

Below we can see the goals we set at the beginning of the project, together with an appreciation on their success:

- **Designing a story generator capable of creating a wide set of short and different stories.**
  We have completed this goal successfully, as we are able to generate a set of 40.460 different short stories.
- **Generating a model of story interest that gives as result the most interesting stories in the provided set.**
  We have completed this goal successfully, as our model has a hit rate of 73.3333 % (which, presumably, will improve with the interaction with end users).
- **Making the system capable of feeding back, improving the model with each interaction.**
  We have completed this goal partially as, although our system is capable of feeding back to improve the model, the intervention of a real user is mandatory to include the story with its new value in the training set.

As a final conclusion, we have to highlight that the good performance of this prototype opens up the possibility of creating more complex generators, based on the methodology and paradigm that we expose.

6.2. Conclusiones

Mientras no seamos capaces de modelar al completo el cerebro humano, no seremos capaces de reproducir cómo funciona. Esto es especialmente crítico en ámbitos como la creatividad, en los que no estamos seguros de cómo o porqué actuamos de la manera en que lo hacemos.

No obstante, nos es posible generar prototipos muy acotados en el ámbito, que nos ayudan a comprender algunos de los procesos que tienen lugar en nuestro cerebro. Con los conocimientos de que disponemos, nos es posible enseñar a un ordenador a “inventar”, basándonos en nuestro caso en la generación automática de historias. El hecho de que

la creatividad sea un concepto tan vago hace que tengamos múltiples líneas de investigación sólo en cuanto a la generación de historias, como representación de sentimientos, relaciones, interacciones entre personajes y entorno, y un largo etcétera.

Habiendo trabajado ya en el ámbito de la generación de historias, comprobamos las ventajas y desventajas de este nuevo enfoque en relación con el anteriormente estudiado (generación basada en agentes).

En cuanto a la generación de historias, consideramos mucho más interesante la generación basada en agentes, ya que en ella son los mismos personajes los que construyen la historia en relación con sus interacciones y objetivos propios, lo que hace que las historias sean mucho más naturales. Sin embargo, para nuestro caso es necesario conocer un conjunto completo de las posibles soluciones, por lo que este enfoque no es posible. Además, una desventaja importante respecto a la generación de gramáticas es el tiempo que tardamos en generar una historia. La interacción entre agentes software autónomos es un proceso pesado computacionalmente, mientras que en las gramáticas "sólo" es necesario generar el árbol completo de soluciones. Este árbol está cerrado, al contrario que con los agentes, que utilizan la información que obtienen (objetivos propios, objetivos comunes, estado del entorno, estado del resto de agentes, etc.) para la generación.

Por esta razón, consideramos la parte de aprendizaje máquina como la más interesante de este sistema. Académicamente, la posibilidad de arrojar luz sobre un tema sobre el que se ha discutido durante largo tiempo (qué elementos hacen interesante a una historia) sin llegar a un consenso resulta muy motivador. Por otro lado, el aprendizaje máquina es un campo que está en auge hoy en día, y que está llamado a tener cada vez más importancia en el futuro gracias a las nuevas tecnologías, como el Big Data. Hemos podido comprobar la gran cantidad de posibilidades y algoritmos que nos brinda este campo de estudio, y el impacto que su elección tiene en los resultados finales.

En cuanto a las features a escoger en un conjunto de datos, nos hemos dado cuenta de que su correcta elección puede suponer un campo de estudio en sí mismo, ya que actualmente es casi una tarea artesanal. De hecho, es digno de mención el hecho de que es un factor diferenciador a la hora de escoger un data scientist u otro por parte de una empresa.

A continuación podemos ver los objetivos que nos marcamos al comienzo del proyecto, junto con una valoración sobre si se ha cumplido o no:

- **Diseñar un generador de historias capaz de crear historias extensas y diferentes.**
  Este objetivo se alteró durante el transcurso del proyecto, pasando a generar historias cortas y diferentes, pero en alto número. Hemos cumplido este objetivo completamente, ya que la cantidad de historias diferentes que podemos generar es de 40.460.
- **Diseñar un modelo de historia interesante que dé como resultado las historias más interesantes del conjunto de historias proporcionadas.**

Hemos cumplido este objetivo completamente, en el sentido de que hemos entrenado un modelo de interés que filtra las historias en función de sus predicciones, utilizando datos proporcionados por usuarios reales. Este modelo tiene un porcentaje de acierto del 73.3333 % (mejorará con la interacción de usuarios reales con el sistema).

- **Conseguir que el sistema sea capaz de realimentarse, mejorando el modelo de filtrado con cada ejecución.**
  Hemos cumplido este objetivo parcialmente ya que, aunque el sistema es capaz de realimentarse para mejorar el modelo, es necesaria la intervención de un usuario real para incluir una historia con una nueva valoración en el conjunto de entrenamiento.

Como conclusión final, debemos destacar que el buen funcionamiento de este prototipo nos abre la posibilidad de crear generadores de historias más complejos, basándonos en la metodología y paradigma (clasificación de historias mediante aprendizaje máquina) aplicados al mismo.

6.3. Trabajo futuro

Aunque hemos completado satisfactoriamente este proyecto, en el sentido de que hemos cumplido con todos los objetivos que nos marcamos inicialmente, hay muchas cosas que mejorar. Además, existen gran cantidad de posibilidades que consideramos sería muy interesante añadir, aunque no hayamos podido hacerlo por falta de tiempo.

A continuación, mostramos algunos de los caminos por los que, de haber dispuesto de más tiempo, nos habría gustado continuar aunque, como hemos dicho anteriormente, las posibilidades en esta rama de investigación son infinitas. Evidentemente, la ampliación directa de cualquiera de las partes del generador (como por ejemplo, más posibilidades en cuanto al clima, o más hechos espontáneos) añadiría variedad y posibilidades a las historias, por lo que hemos decidido obviar estos puntos en los siguientes párrafos.

6.3.1. Ampliación de métricas y conjuntos de entrenamiento

Debido a la falta de tiempo y recursos, el conjunto de entrenamiento utilizado constaba solamente de 269 historias. Para conseguir un modelo más preciso, sería recomendable aumentar dicho conjunto.

Por esa misma razón, las métricas utilizadas fueron en gran parte las ya implementadas para el proyecto WHIM, modificadas para que se ajustaran a nuestras necesidades. Por ello, es posible que algunas de las métricas más fuertemente relacionadas con el interés de las historias no se hayan incluido en la versión final. Por ello, consideramos de gran interés aumentar el número de métricas calculadas, debido a que la inclusión de atributos fuertemente relacionados con el interés mejoraría los resultados del modelo.

Además, aportaría información en el campo de estudios literarios acerca de los elementos que otorgan interés a las historias.

6.3.2. Generador externo de lenguaje natural

Por simplicidad hemos decidido utilizar plantillas sencillas para generar el texto de las historias, a fin de reducir el rechazo del usuario. Sin embargo, sería interesante añadir un generador de lenguaje natural que las transforme en texto narrativo, como por ejemplo el generador de texto TAP (Gervás, 2007).

Consideramos que esta posibilidad sería muy interesante ya que nos permitirá realizar estudios sobre el estilo del texto. De esta forma, podríamos incluir en nuestro modelo de interés features relacionados con el estilo del texto, como por ejemplo el hecho de que el texto esté en presente o pasado, o que la narración sea en primera o tercera persona, lo cual creemos que mejoraría considerablemente el valor de los resultados obtenidos. Por otro lado, el hecho de que sea un generador automático de texto nos da la posibilidad de generar un conjunto completo de historias traducidas en lenguaje natural, lo cual es necesario para extraer conclusiones correctamente, aunque podría hacerse una primera prueba de concepto realizando las traducciones a mano. Como ventaja adicional, presentar las historias como texto normal haría que los usuarios finales se mostraran más receptivos, ya que evitamos las faltas de ortografía o frases mal construidas propias de la generación con plantillas, por lo que podríamos recopilar mayor cantidad de datos en menos tiempo.

6.3.3. Análisis del lenguaje utilizado

Para simplificar el proceso de aprendizaje obtuvimos un modelo de historia interesante que no tiene en cuenta parámetros relacionados con el estilo del lenguaje utilizado para la narración, por lo que no sabemos en qué medida el uso de ciertos recursos literarios puede aumentar el interés.

Por ello (en relación con el apartado anterior) consideramos interesante la posibilidad de añadir el análisis del texto al modelo calculado, obteniendo así una información mucho más valiosa de las valoraciones reales, y mejorando el clasificador.

6.3.4. Cambio del protagonista

Actualmente, por sencillez, el sistema considera como personajes aquellos que aparezcan etiquetados como animales en el conocimiento de dominio. Esto supone una clara limitación, ya que esto sólo es habitual en ciertos ámbitos, como el cine infantil.

Añadiendo nuevo conocimiento de dominio, sería posible hacer que el sistema reconozca los nombres propios como personajes, haciendo que el cambio de contexto mantuviese

la coherencia en mayor medida. Por otro lado, sería necesario modificar algunos métodos de la base del conocimiento, aunque los cambios en el código serían mínimos, y no demasiado complicados. Por ello, consideramos esta ampliación como una de las más sencillas de implementar, que además añadiría mucho valor al sistema.

6.3.5. Ampliación del uso de acciones del servicio web

En el estado actual, el sistema posee un reducido número de acciones predefinidas en la base de conocimiento del sistema, que se usan para obtener nuevas acciones del servicio web de la universidad de Cambridge. Debido a su alto coste en tiempo, esta consulta sólo se realiza una vez por cada historia, generando una nueva por cada acción diferente que el servicio web nos proporciona como respuesta. No obstante, es posible realizar algunas mejoras a este respecto.

En primer lugar, sería interesante utilizar los hechos propios de la mini narrative para obtener acciones. Esto no se ha realizado por simplicidad, pero aumentaría la variedad de las historias. Gracias a que la mini narrative es la misma para todo el conjunto de historias, sería interesante, para cada hecho o acción de la mini narrative, obtener las posibles acciones del servicio web antes de la generación de las historias propiamente dicha. Esto mejoraría el coste temporal de utilizar acciones del servicio web, ya que extraemos la parte costosa de la inclusión de acciones del proceso de generación, debido a que durante el proceso de backtracking sólo tenemos que consultar la base de conocimiento local. Esto nos permitiría añadir acciones obtenidas desde el servicio web en cualquier fase de la historia. También se podría aplicar esto a la obtención de acciones a partir de las acciones de la base de conocimiento, ya que están predefinidas en el sistema, aunque esto último no se implementó por falta de tiempo.

Por último, sería interesante incluir varias acciones seguidas en la misma historia, aunque en este caso el precálculo de esta segunda acción sería mucho más costoso, y posiblemente deba incluirse la consulta durante la generación de la historia. Sin embargo, esto es lo que hacemos actualmente, por lo que consideramos que el resultado sería similar al actual en cuanto a tiempo, aunque con un número mucho mayor de historias que el sistema puede generar, con mayor variedad a su vez.

6.3.6. Generación en función de los hechos

Actualmente en nuestras historias, los hechos, las consecuencias de las relaciones y las acciones en sí mismas no tienen efecto sobre la historia. Por ejemplo, el hecho de que comience una pelea no condiciona las acciones que ocurrirán a continuación, por lo que algunas de nuestras historias no tienen sentido desde el punto de vista de la lógica (si un perro y un caballo se pelean, no es lógico que después el perro monte a lomos del caballo, a no ser que ocurra algo entre ambos hechos que lo justifique).

Esta mejora atacaría directamente una de las principales desventajas de la generación basada en gramáticas formales que es, como exponemos en el apartado 3.2 de este documento, el hecho de que los elementos de la historia funcionan como piezas que se intercambian inopinadamente. De esta forma, el interés general del conjunto de historias se vería incrementado en gran medida, aunque es posible que el número de historias que podemos generar baje.

6.3.7. Cambio de paradigmas

Aunque la generación de historias basada en gramáticas formales se ha mostrado capaz de ofrecernos múltiples posibilidades, creemos que existen enfoques más interesantes a tener en cuenta, como hemos expuesto en las conclusiones. Por ello, consideramos que sería interesante probar diferentes posibilidades de generadores, como la generación basada en agentes o basada en planificación. Otra posibilidad sería una aproximación mixta, en la que las historias siguen estando completamente estructuradas, mientras que las acciones de los personajes se simulan mediante otro paradigma, siendo el ideal el de la interacción entre agentes. De esta forma, conseguimos ventajas presentes en ambos mundos, ya que reducimos en gran medida la carga computacional del uso de agentes, conservando la capacidad de crear conjuntos muy grandes de historias cortas.

Por otro lado, en el presente proyecto sólo se han utilizado dos modelos de aprendizaje máquina: Support Vector Machines y Random Forest. Una posibilidad interesante es la de comprobar el índice de acierto del sistema utilizando diferentes algoritmos. Por cómo está estructurado el código del generador de modelos, nos es posible probar diferentes algoritmos añadiendo pocas líneas de código, sin modificar nada de lo ya escrito.

6.3.8. Base de datos externa

Una de las funcionalidades más interesantes de este proyecto es la capacidad de realimentarse, es decir, la posibilidad de utilizar el feedback de los usuarios para mejorar el modelo. Sin embargo, actualmente es un proceso que se realiza individualmente con cada usuario, de forma que cada copia del sistema recibirá diferente feedback.

Por ello, consideramos que sería muy interesante la centralización del proceso de realimentación. Utilizando un servidor, recibiremos el feedback de los usuarios, guardando aquellas historias cuya etiqueta haya cambiado. De esta forma, podemos llevar la cuenta de cuántos usuarios consideran que dicha etiqueta debe cambiar, de forma que evitamos la introducción de anomalías, es decir, usuarios que consideran interesantes historias que para la mayoría no lo son.

La otra ventaja inherente a este cambio es la centralización del conocimiento. Gracias a ello, podemos hacer que el conocimiento global esté centralizado en nuestro servidor, de forma que la aplicación simplemente tendría que descargarlo para usarlo. Así, el

funcionamiento para diferentes instancias de la aplicación es el mismo, con una mejora mucho más rápida del modelo.

Apéndice A

Gramática utilizada

:- module(grammar, [story/8]).

:- use_module(kb).

story(WV, MN, MNO, St, StOut, C) –>restart, beginning(WV, MN, MNB, St, StB, C1), development(WV, MNB, MND, StB, StD, C2), outcome(WV, MND, MNO, StD, StOut, C3), { C is C1 * C2 * C3 }.

/* Beggining ———————————————— */

beginning(WV, [NP | MN], MN, St, StOut, C) –>NP, {foldl(applyfact(WV), NP, (St, 1.0), (StM, C)), introduce_main_characters(NP, CF)}, CF, {append(StM, CF, StOut)}.

beginning(WV, [NP | MN], MN, St, StOut, C) –>{frase_trivial(F)}, F, NP, {foldl(applyfact(WV), NP, (St, 1.0), (StMid, C)), introduce_main_characters(NP, CF)}, CF, {append(F, CF, Mid), append(Mid, StMid, StOut)}.

/* Development ———————————————— */

development(_, [NP | MN], MN, St, StOut, 1.0) –>NP, {introduce_secondary_characters(St, NP, CF), append(St, CF, StOut)}.

development(WV, [NP | MN], MN, St, StOut, C) –>NP, {happen(WV, Ef, St, NP, C), introduce_secondary_characters(St, NP, CF)}, Ef, {append(St, Ef, Mid), append(Mid, CF, StOut)}.

development(WV, [NP | MN], MN, St, StOut, C) –>NP, {step(WV, St, StMid, Ev1, C1), fetch_actions(Ev1, AcJSON), choose_one_action(StMid, AcJSON, [score(S), Ev2]), introduce_secondary_characters(St, [Ev1, Ev2], CF)}, [Ev1], [Ev2], CF, {append(St, [Ev1], Mid), append(Mid, [Ev2], Mid2), append(Mid2, CF, StOut)}, {C is C1 * S}.

/* Outcome ———————————————— */

outcome(_, [NP | MN], MN, St, St, 1.0) –>NP.

restart –>{retract_all}.

Apéndice B

Plantillas de lenguaje natural

```
/* GENERA TEXTO PARA LAS HISTORIAS */

textualizeStories(FileName) :- prologize(FileName, TmpFile), readf(TmpFile, Stories), takeEndOfFile(Stories, SEnd), stories2text(SEnd, TxtSt), !, atom_concat(FileName, '_textualized.txt', File), open(File, write, Str), maplist(writeln(Str), TxtSt), close(Str), delete_file(TmpFile).

/* PLANTILLAS */

stories2text([(St, _)], [Txt]) :- convert_to_text(St, Txt). stories2text([(St, _) | RS], Frs) :- convert_to_text(St, Txt), stories2text(RS, RFStr), append([Txt], RFStr, Frs).

convert_to_text([F], Txt) :- textualize(F, StTxt), atom_concat(StTxt, 'The end.', Txt).


convert_to_text([[X, sky], [Y, wind], [Z, temperature] | RF], Frs) :- text_trivial_phrase([[X, sky], [Y, wind], [Z, temperature]], Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int), atom_concat(Int, RFrs, Frs), !.


convert_to_text([[X, sky], [not, [Y, wind]], [Z, temperature] | RF], Frs) :- text_trivial_phrase([[X, sky], [not, [Y, wind]], [Z, temperature]], Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int), atom_concat(Int, RFrs, Frs), !.


convert_to_text([[main_character, X], [main_character, Y] | RF], Frs) :- text_main_characters([[m
X], [main_character, Y]], Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int5), atom_concat(Int5, RFrs, Frs), !.


convert_to_text([[Rel, X, Y], [Eff, Z, W] | RF], Frs) :- is_relation(Rel), is_effect(Eff), textualize([Rel, X, Y], TxtR), textualize([Eff, Z, W], TxtEf), atom_concat(TxtR, ', so ', Int), atom_concat(Int, TxtEf, Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int5), atom_concat(Int5, RFrs, Frs), !.
```

```
convert_to_text([[capableof, dog, ride_horse], [doesfor, dog, ride_horse, fun]], Frs) :- textualize([capableof, dog, ride_horse], TxtP), textualize([doesfor, dog, ride_horse, fun], TxtEf), downcase_atom(TxtEf, LCTxt), atom_concat(TxtP, ', so, instead, ', Int), atom_concat(Int, LCTxt, Str), atom_concat(Str, '. The end.', Frs), !.

convert_to_text([[hasproperty, dog, old], [not, [capableof, dog, run]] | RF], Frs) :- textualize([hasproperty, dog, old], TxtP), textualize([not, [capableof, dog, run]], TxtEf), atom_concat(TxtP, ', so he ', Int), atom_concat('the dog ', TxtEfD, TxtEf), atom_concat(Int, TxtEfD, Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int5), atom_concat(Int5, RFrs, Frs), !.

convert_to_text([[date | RD], [time | RT] | RF], Frs) :- textualize([date | RD], TxtDate), textualize([time | RT], TxtTime), downcase_atom(TxtTime, LCTime), atom_concat(TxtDate, ', and ', Int), atom_concat(Int, LCTime, Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int5), atom_concat(Int5, RFrs, Frs), !.

convert_to_text([[isa | R], [doesfor, dog, run, fun] | RF], Frs) :- textualize([isa | R], TxtDog), textualize([doesfor, dog, run, fun], TxtDoes), atom_concat('The dog ', TxtAct, TxtDoes), atom_concat(TxtDog, ', and he ', Int), atom_concat(Int, TxtAct, Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int5), atom_concat(Int5, RFrs, Frs), !.

convert_to_text([F | RF], Frs) :- textualize(F, Txt), convert_to_text(RF, RFrs), atom_concat(Txt, '. ', Int), atom_concat(Int, RFrs, Frs).

is_relation(brother). is_relation(friend). is_relation(enemy). is_relation(parent).

is_effect(love). is_effect(hate). is_effect(hangout).

day(1, 'Monday'). day(2, 'Tuesday'). day(3, 'Wednesday'). day(4, 'Thursday'). day(5, 'Friday'). day(6, 'Saturday'). day(7, 'Sunday').

month(1, 'January'). month(2, 'February'). month(3, 'March'). month(4, 'April'). month(5, 'May'). month(6, 'June'). month(7, 'July'). month(8, 'August'). month(9, 'September'). month(10,'October'). month(11, 'November'). month(12, 'December').

text_trivial_phrase([Sky, Wind, Tmp], Txt) :- textualize(Sky, TxtSky), textualize(Wind, TxtW), textualize(Tmp, TxtTemp), atom_concat(TxtSky, ', ', Int1), atom_concat(Int1, TxtW, Int2), atom_concat(Int2, ' and ', Int3), atom_concat(Int3, TxtTemp, Txt).

text_main_characters([[main_character, X], [main_character, Y]], Txt) :- atom_concat('The ', X, Int), atom_concat(Int, ' and ', Int2), atom_concat(Int2, 'the ', Int3), atom_concat(Int3, Y, Int4), atom_concat(Int4, ' were the main characters', Txt).
```

textualize([isa, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' was an ', In), atom_concat(In, Y, N), !.

textualize([doesfor, X, Y, Z], N) :- atom_concat('The ', X, In), atom_concat(In, ' ', Inner), (translate(do, Y, T), atom_concat(Inner, T, In2) ; atom_concat(Inner, Y, Int), atom_concat(Int, 's', In2)), atom_concat(In2, ' for ', In3), atom_concat(In3, Z, N), !.

textualize([capableof, X, ride_horse], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' was capable of ', In), atom_concat(In, 'riding a horse', N), !.

textualize([capableof, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' was capable of ', In), atom_concat(In, Y, N), !.

textualize([main_character, X], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' was a main character', N), !.

textualize([hasproperty, dog, old], N) :- N = 'The dog was old', !.

textualize([hasproperty, X, Y], N) :- atom_concat(X, ' had the property of ', In), atom_concat(In, Y, N), !.

textualize([learnto, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' learned to ', In), (translate(learn, Y, T), atom_concat(In, T, N) ; atom_concat(In, Y, N)), !.

textualize([event, X], N) :- atom(X), atom_concat('The event ', X, In), atom_concat(In, ' occurred', N), !.

textualize([event, X], N) :- format(atom(A), -w", X), atom_concat('The event ', A, In), atom_concat(In, ' occurred', N), !.

textualize([secondary_character, X], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' was a secondary character', N), !.

textualize([brother, X, Y], N) :- atom_concat('The ', Y, Int), atom_concat(Int, ' was ', In), atom_concat(In, X, In2), atom_concat(In2, 's brother', N), !.

textualize([friend, X, Y], N) :- atom_concat('The ', Y, Int), atom_concat(Int, ' was ', In), atom_concat(In, X, In2), atom_concat(In2, 's friend', N), !.

textualize([enemy, X, Y], N) :- atom_concat('The ', Y, Int), atom_concat(Int, ' was ', In), atom_concat(In, X, In2), atom_concat(In2, 's enemy', N), !.

textualize([parent, X, Y], N) :- atom_concat('The ', Y, Int), atom_concat(Int, ' was ', In), atom_concat(In, X, In2), atom_concat(In2, 's parent', N), !.

textualize([love, X, Y], N) :- atom_concat('the ', X, Int), atom_concat(Int, ' loved the ', In), atom_concat(In, Y, N), !.

textualize([hate, X, Y], N) :- atom_concat('the ', X, Int), atom_concat(Int, ' hated the ', In), atom_concat(In, Y, N), !.

textualize([hangout, X, Y], N) :- atom_concat('the ', X, Int), atom_concat(Int, ' hanged out with the ', In), atom_concat(In, Y, N), !.

textualize([start, rain], 'It stareds to rain').

textualize([start, fight], 'A fight started').

textualize([sing, people], 'Everybody started to sing').

textualize([get, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' got ', In), atom_concat(In, Y, N), !.

textualize([use, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' used ', In), atom_concat(In, Y, N), !.

textualize([sing, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' sang ', In), atom_concat(In, Y, N), !.

textualize([preparefor, X, Y], N) :- atom_concat('The ', X, Int), atom_concat(Int, ' prepared for ', In), atom_concat(In, Y, N), !.
textualize([X, sky], N) :- atom_concat('The sky was ', X, N), !.

```
    textualize([X, wind], N) :- atom_concat('the wind was ', X, N), !.
textualize([X, temperature], N) :- atom_concat('the temperature was ', X, N), !.
textualize([date, X, Y, Z], N) :- day_of_the_week(date(X, Y, Z), DayNum), day(DayNum,
Day), atom_concat('It was ', Day, Int1), atom_concat(Int1, ', the ', Int2), atom_concat(Int2,
Z, Int3), atom_concat(Int3, ' of ', Int4), month(Y, Month), atom_concat(Int4, Month,
Int5), atom_concat(Int5, ', ', Int6), atom_concat(Int6, X, N), !.


    textualize([time, X, Y], N) :- atom_concat('The time was ', X, In), atom_concat(In,
':', In3), atom_concat(In3, Y, N), !.


    /* NOT */


    textualize([not, [X, wind]], N) :- atom_concat('the wind was not ', X, N), !.
textualize([not, [capableof, X, ride_horse]], N) :- atom_concat('The ', X, Int), atom_concat(Int,
' was not capable of ', In), atom_concat(In, 'riding a horse', N), !.
textualize([not, [capableof, X, Y]], N) :- atom_concat('the ', X, Int), atom_concat(Int, '
was not capable of ', In), (translate(capable, Y, T), atom_concat(In, T, N) ; atom_concat(In,
Y, N)), !.


    textualize([not, X], N) :- textualize(X, F), atom_concat('not ', F, N), !.
/*_____________________________________________*/
translate(learn, ride_horse, 'ride a horse').
translate(capable, ride_horse, 'riding a horse').
translate(do, ride_horse, 'rode a horse').


    translate(do, run, 'ran').
translate(capable, run, 'running').


    getFirstNElements(Src, N, L) :- findall(E, (nth1(I,Src,E), I =<N), L).
```

Bibliografía

Black, John B. y Robert Wilensky (1979). «An evaluation of story grammars». En: *Cognitive Science* 3.3, págs. 213-229. ISSN: 03640213.

Bundy, Alan (2007). «Cooperating reasoning processes: More than just the sum of their parts». En: *IJCAI International Joint Conference on Artificial Intelligence*, págs. 2-11. ISBN: 10450823 (ISSN).

Chang, Chih-chung y Chih-jen Lin (2011). «LIBSVM : A Library for Support Vector Machines». En: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, págs. 1-39. ISSN: 21576904.

Chen, Pai-Hsuen, Chih-Jen Lin y B. Schölkopf (2005). «A tutorial on nu-support vector machines». En: *Applied Stochastic Models in Business and Industry* 21.2, págs. 111-136. ISSN: 1524-1904.

Chomsky, Noam (1956). «Three models for the description of language». En: *IRE Transactions on Information Theory* 2.3, págs. 113-124. ISSN: 0096-1000.

— (1972). *Studies on semantics in generative grammar*. Vol. 107, pág. 103. ISBN: 9027979642.

Colton, Simon (2012). «The painting fool: Stories from building an automated painter». En: *Computers and Creativity*. Vol. 9783642317279, págs. 3-38. ISBN: 9783642317279.

Colton, Simon y Geraint A. Wiggins (2012). «Computational creativity: The final frontier?» En: *Frontiers in Artificial Intelligence and Applications*. Vol. 242, págs. 21-26. ISBN: 9781614990970.

Cortes, Corinna y Vladimir Vapnik (1995). «Support-Vector Networks». En: *Machine Learning* 20.3, págs. 273-297. ISSN: 15730565.

Dehn, Natalie (1981). «Story Generation After TALE-SPIN». En: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, págs. 16-18.

Eigenfeldt, Arne, Adam Burnett y Philippe Pasquier (2012). «Evaluating Musical Metacreation in a Live Performance Context». En: *Proceedings of the Third International Conference on Computational Creativity (ICCC 2012)*, págs. 140-144.

Gervás, Pablo (2007). *TAP: a Text Arranging Pipeline*. Inf. téc. Natural Interaction based on Language Group Technical Report. Universidad Complutense de Madrid.

— (2009). «Computational Approaches to Storytelling and Creativity». En: *AI Magazine* 30, pág. 49. ISSN: 0738-4602.

— (2013). «Propp's Morphology of the Folk Tale as a Grammar for Generation». En: *Proceedings of the 4th Workshop on Computational Models of Narrative (CMN'13)*. Vol. 32, págs. 106-122. ISBN: 9783939897576.

Grasbon, Dieter y Norbert Braun (2001). «A morphological approach to interactive storytelling». En: *Proc. CAST01, Living in Mixed Realities. Special ...* Págs. 337-340.

Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann y Ian H Witten (2009). «The WEKA data mining software». En: *SIGKDD Explorations Newsletter* 11.1, pág. 10. ISSN: 19310145.

Klein, Sheldon, John F. Aeschlimann, David F. Balsinger, Steven L. Converse, Claudine Court, Mark Foster, Robin Lao, John D. Oakley y Joel Smith (1973). «Automatic novel writing: A status report». En: *Technical report 186*.

Kolodner, J. L. (1980). «Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model». Tesis doct. Yale University.

Laclaustra, Iván M., José L. Ledesma, Gonzalo Méndez y Pablo Gervás (2014). «Kill the Dragon and Rescue the Princess: Designing a Plan-based Multi-agent Story Generator». En: *5th International Conference on Computational Creativity*.

Lang, Raymond (1997). «A Formal Model for Simple Narratives». Tesis doct. Tulane University.

Lebowitz, Michael (1983). «Creating a Story-Telling Universe». En: *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, págs. 63-65. ISBN: 0865760640.

León, Carlos (2011). «Stella - A Story Generation System for Generic Scenarios». En: *Proceedings of the Second International Conference on Computational Creativity*, pág. 162. ISBN: 9786074774870.

— (2014). «Creativity in Story Generation From the Ground Up: Non-deterministic Simulation driven by Narrative». En: *5th International Conference on Computational Creativity, ICCC. Ljubljana, Slovenia*.

León, Carlos y Pablo Gervás (2010). «The role of evaluation-driven rejection in the successful exploration of a conceptual space of stories». En: *Minds and Machines* 20.4, págs. 615-634. ISSN: 09246495.

Maletic, Jonathan I y Andrian Marcus (2010). «Data Mining and Knowledge Discovery Handbook». En: *Data Mining and Knowledge Discovery Handbook* Mlc, pág. 1306. ISSN: 14337851.

McDaniel, Mark a., Paula J. Waddill, Kraig Finstad y Tammy Bourg (2000). «The effects of text-based interest on attention and recall.» En: *Journal of Educational Psychology* 92.3, págs. 492-502. ISSN: 0022-0663.

Meehan, Jr (1977). «TALE-SPIN, An Interactive Program that Writes Stories.» En: *Ijcai*, págs. 91-98.

Ng, H. K. Tony (2006). «Statistics: An Introduction Using R». En: *Technometrics* 48, págs. 308-308. ISSN: 0040-1706.

Pachet, François (2002). «The Continuator: Musical Interaction With Style». En: *Proceedings of the International Computer Music Conference* September, págs. 333-341. ISSN: 0929-8215.

Pérez ý Pérez, Mike Sharples, Rafael (2001). «MEXICA: A computer model of a cognitive account of creative writing». En: *Journal of Experimental & Theoretical Artificial Intelligence* 13, págs. 119-139. ISSN: 0952813X.

Propp, V (1968). «Morphology of the folktale». En: *Publications of the American Folklore Society Bibliographical and special series* 9, xxvi, 158 p. ISSN: 1469-994X.

Real Academia Española (2014). *DICCIONARIO DE LA LENGUA ESPAÑOLA*.

Riedl, Mark O. y R. Michael Young (2010). «Narrative planning: Balancing plot and character». En: *Journal of Artificial Intelligence Research* 39, págs. 217-268. ISSN: 10769757.

Sadoski, Mark, Ernest Goetz, Arturo Olivarez, Sharon Lee y Nancy Roberts (1990). «Imagination in story reading: The role of imagery, verbal recall, story analysis, and processing levels». En: *Journal of Literacy Research* 22.1, págs. 55-70. ISSN: 1086-296X.

Samuel, A. L. (1959). «Some Studies in Machine Learning Using the Game of Checkers». En: *IBM Journal of Research and Development* 3.3, págs. 210-229. ISSN: 0018-8646.

Schank, R. C. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University.

Schaper, Eva (1978). «Fiction and the suspension of disbelief». En: *The British Journal of Aesthetics* 18.1, págs. 31-44. ISSN: 00070904.

Shalev-Shwartz, Shai y Shai Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Sharples, Mike (1999). *How We Write: Writing as Creative Design*. London: Routledge.

Simon, Phil. Wiley and (2013). «Too Big to Ignore : The Business Case for Big Data». En: *Journal of Chemical Information and Modeling* 53.9, págs. 1689-1699. ISSN: 1098-6596.

Theune, Mariet, Sander Faas, Er Faas, Anton Nijholt y Dirk Heylen (2003). «The Virtual Storyteller: Story Creation by Intelligent Agents». En: *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, págs. 204-215.

Turner, Scott R (1993). *MINSTREL: A computer model of creativity and storytelling*.

Wooldridge, Michael (2002). «Introduction to Multiagent Systems». En: *Information Retrieval* 30, págs. 152-183. ISSN: 03713611.

Young, Raymond De y Martha C. Monroe (1996). «Some Fundamentals of Engaging Stories». En: *Environmental Education Research* 2.2, págs. 171-187. ISSN: 1350-4622.